



# Windchill<sup>®</sup> Customizer's Guide

Windchill 7.0  
December 2003

## **Copyright © 2003 Parametric Technology Corporation. All Rights Reserved.**

User and training documentation from Parametric Technology Corporation (PTC) is subject to the copyright laws of the United States and other countries and is provided under a license agreement that restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the PTC copyright notice and any other proprietary notice provided by PTC. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of PTC and no authorization is granted to make copies for such purposes.

Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from PTC. UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION.

## **Registered Trademarks of Parametric Technology Corporation or a Subsidiary**

Advanced Surface Design, Behavioral Modeling, CADD5, Computervision, EPD, EPD.Connect, Expert Machinist, Flexible Engineering, HARNESSDESIGN, Info\*Engine, InPart, MECHANICA, Optegra, Parametric Technology, Parametric Technology Corporation, PHOTORENDER, Pro/DESKTOP, Pro/E, Pro/ENGINEER, Pro/HELP, Pro/INTRALINK, Pro/MECHANICA, Pro/TOOLKIT, PTC, PT/Products, Shaping Innovation, and Windchill.

## **Trademarks of Parametric Technology Corporation or a Subsidiary**

3DPAINT, Associative Topology Bus, AutobuildZ, CDRS, CounterPart, Create Collaborate Control, CV, CVact, CVaec, CVdesign, CV-DORS, CVMAC, CVNC, CVToolmaker, DataDoctor, DesignSuite, DIMENSION III, DIVISION, e/ENGINEER, eNC Explorer, Expert MoldBase, Expert Toolmaker, GRANITE, ISSM, KDiP, Knowledge Discipline in Practice, Knowledge System Driver, ModelCHECK, MoldShop, NC Builder, PartSpeak, Pro/ANIMATE, Pro/ASSEMBLY, Pro/CABLING, Pro/CASTING, Pro/CDT, Pro/CMM, Pro/COLLABORATE, Pro/COMPOSITE, Pro/CONCEPT, Pro/CONVERT, Pro/DATA for PDGS, Pro/DESIGNER, Pro/DETAIL, Pro/DIAGRAM, Pro/DIEFACE, Pro/DRAW, Pro/ECAD, Pro/ENGINE, Pro/FEATURE, Pro/FEM-POST, Pro/FICIENCY, Pro/FLY-THROUGH, Pro/HARNESS, Pro/INTERFACE, Pro/LANGUAGE, Pro/LEGACY, Pro/LIBRARYACCESS, Pro/MESH, Pro/Model.View, Pro/MOLDESIGN, Pro/NC-ADVANCED, Pro/NC-CHECK, Pro/NC-MILL, Pro/NCPOST, Pro/NC-SHEETMETAL, Pro/NC-TURN, Pro/NC-WEDM, Pro/NC-Wire EDM, Pro/NETWORK ANIMATOR, Pro/NOTEBOOK, Pro/PDM, Pro/PHOTORENDER, Pro/PIPING, Pro/PLASTIC ADVISOR, Pro/PLOT, Pro/POWER DESIGN, Pro/PROCESS, Pro/REPORT, Pro/REVIEW, Pro/SCAN-TOOLS, Pro/SHEETMETAL, Pro/SURFACE, Pro/VERIFY, Pro/Web.Link, Pro/Web.Publish, Pro/WELDING, Product Development Means Business, Product First, ProductView, PTC Precision, Shrinkwrap, Simple Powerful Connected, The Product Development Company, The Way to Product First, Wildfire, Windchill DynamicDesignLink, Windchill PartsLink, Windchill PDMLink, Windchill ProjectLink, and Windchill SupplyLink.

## **Third-Party Trademarks**

Adobe is a registered trademark of Adobe Systems. Advanced ClusterProven, ClusterProven, and the ClusterProven design are trademarks or registered trademarks of International Business Machines Corporation in the United States and other countries and are used under license. IBM Corporation does not warrant and is not responsible for the operation of this software product. AIX is a registered trademark of IBM Corporation. Allegro, Cadence, and Concept are registered trademarks of Cadence Design Systems, Inc. AutoCAD and

AutoDesk Inventor are registered trademarks of Autodesk, Inc. Baan is a registered trademark of Baan Company. CADAM and CATIA are registered trademarks of Dassault Systemes. COACH is a trademark of CADTRAIN, Inc. DOORS is a registered trademark of Telelogic AB. FLEXlm is a registered trademark of GLOBETrotter Software, Inc. Geomagic is a registered trademark of Raindrop Geomagic, Inc. EVERSINC, GROOVE, GROOVEFEST, GROOVE.NET, GROOVE NETWORKS, iGROOVE, PEERWARE, and the interlocking circles logo are trademarks of Groove Networks, Inc. Helix is a trademark of Microcadam, Inc. HOOPS is a trademark of Tech Soft America, Inc. HP-UX is a registered trademark and Tru64 is a trademark of the Hewlett-Packard Company. I-DEAS, Metaphase, Parasolid, SHERPA, Solid Edge, and Unigraphics are trademarks or registered trademarks of Electronic Data Systems Corporation (EDS). InstallShield is a registered trademark and service mark of InstallShield Software Corporation in the United States and/or other countries. Intel is a registered trademark of Intel Corporation. IRIX is a registered trademark of Silicon Graphics, Inc. MatrixOne is a trademark of MatrixOne, Inc. Mentor Graphics and Board Station are registered trademarks and 3D Design, AMPLE, and Design Manager are trademarks of Mentor Graphics Corporation. MEDUSA and STHENO are trademarks of CAD Schroer GmbH. Microsoft, Microsoft Project, Windows, the Windows logo, Windows NT, Visual Basic, and the Visual Basic logo are registered trademarks of Microsoft Corporation in the United States and/or other countries. Netscape and the Netscape N and Ship's Wheel logos are registered trademarks of Netscape Communications Corporation in the U.S. and other countries. Oracle is a registered trademark of Oracle Corporation. OrbixWeb is a registered trademark of IONA Technologies PLC. PDGS is a registered trademark of Ford Motor Company. RAND is a trademark of RAND Worldwide. Rational Rose is a registered trademark of Rational Software Corporation. RetrievalWare is a registered trademark of Convera Corporation. RosettaNet is a trademark and Partner Interface Process and PIP are registered trademarks of "RosettaNet," a nonprofit organization. SAP and R/3 are registered trademarks of SAP AG Germany. SolidWorks is a registered trademark of SolidWorks Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and in other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun logo, Solaris, UltraSPARC, Java and all Java based marks, and "The Network is the Computer" are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and in other countries. VisTools is a trademark of Visual Kinematics, Inc. (VKI). VisualCafé is a trademark of WebGain, Inc. WebEx is a trademark of WebEx Communications, Inc.

### **Licensed Third-Party Technology Information**

Certain PTC software products contain licensed third-party technology: Rational Rose 2000E is copyrighted software of Rational Software Corporation. RetrievalWare is copyrighted software of Convera Corporation. VisualCafé is copyrighted software of WebGain, Inc. VisTools library is copyrighted software of Visual Kinematics, Inc. (VKI) containing confidential trade secret information belonging to VKI. HOOPS graphics system is a proprietary software product of, and is copyrighted by, Tech Soft America, Inc. G-POST is copyrighted software and a registered trademark of Intercim. VERICUT is copyrighted software and a registered trademark of CGTech. Pro/PLASTIC ADVISOR is powered by Moldflow technology. Moldflow is a registered trademark of Moldflow Corporation. The JPEG image output in the Pro/Web.Publish module is based in part on the work of the independent JPEG Group. DFORMD.DLL is copyrighted software from Compaq Computer Corporation and may not be distributed. METIS, developed by George Karypis and Vipin Kumar at the University of Minnesota, can be researched at <http://www.cs.umn.edu/~karypis/metis>. METIS is © 1997 Regents of the University of Minnesota. LightWork Libraries are copyrighted by LightWork Design 1990-2001. Visual Basic for Applications and Internet Explorer is copyrighted software of Microsoft Corporation. Adobe Acrobat Reader is copyrighted software of Adobe Systems. Parasolid © Electronic Data Systems (EDS). Windchill Info\*Engine Server contains IBM XML Parser for Java Edition and the IBM Lotus XSL Edition. Pop-up calendar components Copyright © 1998 Netscape Communications Corporation. All Rights Reserved. TECHNOMATIX is copyrighted software and contains proprietary information of Technomatix Technologies Ltd. Apache Server, Tomcat, Xalan, and Xerces are technologies developed by, and are copyrighted software of, the Apache Software Foundation (<http://www.apache.org/>) - their use is subject to the terms and limitations at: <http://www.apache.org/LICENSE.txt>. UnZip (© 1990-2001 Info-ZIP, All Rights Reserved) is provided "AS IS" and WITHOUT WARRANTY OF ANY KIND. For the complete Info-ZIP license see

<ftp://ftp.info-zip.org/pub/infozip/license.html>. Gecko and Mozilla components are subject to the Mozilla Public License Version 1.1 at <http://www.mozilla.org/MPL/>. Software distributed under the MPL is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the MPL for the specific language governing rights and limitations. Technology "Powered by Groove" is provided by Groove Networks, Inc. Technology "Powered by WebEx" is provided by WebEx Communications, Inc. Acrobat Reader is Copyright © 1998 Adobe Systems Inc. Oracle 8i run-time, Copyright © 2000 Oracle Corporation. The Java™ Telnet Applet (StatusPeer.java, TelnetIO.java, TelnetWrapper.java, TimedOutException.java), Copyright © 1996, 97 Mattias L. Jugel, Marcus Meißner, is redistributed under the [GNU General Public License](#). This license is from the original copyright holder and the Applet is provided WITHOUT WARRANTY OF ANY KIND. You may obtain a copy of the source code for the Applet at <http://www.mud.de/se/jta> (for a charge of no more than the cost of physically performing the source distribution), by sending e-mail to [leo@mud.de](mailto:leo@mud.de) or [marcus@mud.de](mailto:marcus@mud.de)-you are allowed to choose either distribution method. The source code is likewise provided under the [GNU General Public License](#). GTK+The GIMP Toolkit are licensed under the [GNU LPGL](#). You may obtain a copy of the source code at <http://www.gtk.org/>, which is likewise provided under the [GNU LPGL](#). zlib software Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

#### UNITED STATES GOVERNMENT RESTRICTED RIGHTS LEGEND

This document and the software described herein are Commercial Computer Documentation and Software, pursuant to FAR 12.212(a)-(b) (OCT'95) or DFARS 227.7202-1(a) and 227.7202-3(a) (JUN'95), is provided to the US Government under a limited commercial license only. For procurements predating the above clauses, use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 (OCT'88) or Commercial Computer Software-Restricted Rights at FAR 52.227-19(c)(1)-(2) (JUN'87), as applicable. 040103

**Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494 USA**





# Contents

<b>Change Record .....</b>	<b>xvii</b>
<b>About This Guide.....</b>	<b>xix</b>
Related Documentation .....	xix
Technical Support.....	xx
Documentation for PTC Products.....	xx
Comments .....	xx
Documentation Conventions .....	xx
<b>Customizing Modeled Business Objects .....</b>	<b>1-1</b>
Windchill Customization Points .....	1-2
Windchill Supported API .....	1-2
Javadoc .....	1-3
Handling Icons.....	1-3
Creating Large Objects (LOBs) .....	1-5
Modeling Large Objects.....	1-5
DDL Generation.....	1-5
Reading and Writing LOBs .....	1-5
Small BLOBs .....	1-6
Inline BLOBs.....	1-6
Example.....	1-7
<b>Customizing Modeled Elements .....</b>	<b>2-1</b>
Customizing Column Lengths.....	2-2
<b>Customizing Business Logic .....</b>	<b>3-1</b>
Identified Business Classes.....	3-2
Identity Attributes .....	3-2
How Identity Is Assigned and Managed .....	3-2
When to Use the IdentityService .....	3-4
How to Use the IdentityService.....	3-5
How to Implement System-Managed Identity .....	3-15
Customizing Iteration and Version Identifiers.....	3-17
Customizing Indexing Behavior .....	3-19
RetrievalWare Fields .....	3-19

Setting the Length of Path Entries.....	3-21
Creating a Custom RwareIndexDelegate.....	3-22
Information Transfer When Indexing a Windchill Business Object.....	3-26
Customizing the Enterprise Search Interface.....	3-28
Customizing a Bill of Materials.....	3-28
Overview .....	3-28
Customization.....	3-28
Current Implementation of Visitors.....	3-31
Customizing User Preferences .....	3-32
The Preference Hierarchy and Delegates .....	3-32
Writing a Servlet Helper to Use URLFactory Functionality .....	3-36
Example URLFactory Implementation.....	3-37
<b>Customizing GUIs.....</b>	<b>4-1</b>
Customizing Windchill Portal Pages .....	4-2
Customizing Colors in HTML and JSP Clients.....	4-5
Customizing Localizable Display Names .....	4-10
Customizing Local Searches .....	4-12
Adding Your Site Classes.....	4-12
Calling the Search Local Task from Client Code.....	4-14
Changing Date Formats for Search Criteria.....	4-16
Customizing the HTML Search .....	4-17
Customizing the Windchill Explorer .....	4-23
Adding Business Classes to Windchill Explorer .....	4-23
Adding 'Save As' Functionality to a New Document or Part Class.....	4-27
Customizing Objects Created from the Windchill Explorer .....	4-31
Making Parts Content Holders .....	4-32
Adding URLs to the Method Server .....	4-32
Writing Methods That Handle HTTP Requests .....	4-32
Generating HTML Dynamically .....	4-38
<b>Customizing service.properties .....</b>	<b>5-1</b>
Customizing service.properties.....	5-2
<b>Creating Load Methods .....</b>	<b>6-1</b>
Overview of Load Utilities .....	6-2
Customizing Loading .....	6-4
Map File.....	6-4
Data File.....	6-5
Creating New Methods for Loading .....	6-6
Existing Load Methods.....	6-9

<b>Customizing the HTML Client.....</b>	<b>7-1</b>
Overview.....	7-3
Creating a View Page in the HTML Client with an Association—Example 1.....	7-14
Before Starting.....	7-14
Creating the HTML Template File.....	7-14
Registering the HTML Template.....	7-17
Creating the Template Processor.....	7-18
Registering the Template Processor.....	7-20
Adding a Link for Your New Page to the HTML Client.....	7-20
Finding Your New HTML Page.....	7-20
Adding a View of Associations to Your Page.....	7-20
Customizing the View of Associations.....	7-21
How to Implement Template Processors.....	7-23
The HTTPState Class and Template Processing.....	7-23
General Comments on Modeling Files for Template Processing.....	7-26
The Default Implementation.....	7-26
Availability of the Page Depends on State or Permissions.....	7-30
A View Page Requiring Custom Processing of QueryString or POST Data.....	7-34
A View Page Requiring Custom Processing of HTML Template File Name or Path.....	7-36
Supplementary Details of Implementation.....	7-38
How to Use the HTML Template Factory.....	7-41
Overview.....	7-41
Default Usage.....	7-42
Setting the Locale Preference.....	7-45
Overriding the htmltemplate.properties Entry.....	7-45
Overriding the Service Name in htmltemplate.properties.....	7-46
Adding an Action in the HTML Client—Example 2.....	7-47
Before Starting.....	7-47
Creating the ActionDelegate and URLActionDelegate.....	7-47
Registering the ActionDelegate and the URLActionDelegate.....	7-51
Adding a Link to the HTML Client for Your New Page.....	7-52
Creating the HTML Template to Generate the HTML Form.....	7-52
Registering the HTML Template.....	7-55
Creating the Template Processor.....	7-55
Registering the Template Processor.....	7-57
Finding Your New HTML Page.....	7-57
Creating the FormTaskDelegate.....	7-58
Registering the FormTaskDelegate.....	7-59
Finding Your New HTML Page.....	7-60

Form Processing and Action Processing .....	7-61
Overview of Invoking an Action .....	7-61
Overview of Generating and Processing a Form .....	7-61
HTTPState and Form Processing .....	7-62
The invokeAction and processForm Methods .....	7-63
How to Invoke an Action.....	7-64
How to Generate and Process a Form.....	7-67
Properties and Property Files .....	7-71
Application Context Service/Resource Properties.....	7-72
HTML Client Services .....	7-80
SubTemplateProcessing .....	7-80
ProcessorService .....	7-82
Adding an HTML Help Link .....	7-85
Adding Help Using Hyperlinked Text.....	7-85
Adding Help Using a Button .....	7-90
Adding Help Using the Global Navigation Bar.....	7-91
Overview of HTML Components, Table, and Table Service .....	7-92
HTML Components .....	7-93
The HTMLTable .....	7-97
HTML Table Service.....	7-100
Using HTML Components.....	7-102
The Standard Process.....	7-102
Using the HTMLComponentFactory.....	7-103
Creating a New Base HTML Component.....	7-103
Creating a Custom HTML Component .....	7-104
Selecting a New Context(#Service Name) for Initializing the Default Tag Attributes .....	7-109
Printing Tags Only.....	7-110
Printing the Default Tag Values Only .....	7-110
Using HTML Tables and Custom Tables .....	7-111
Printing a Standalone HTML Table .....	7-111
Setting cellSelector to Select Custom HTML Components for the Cells.....	7-113
Enabling or Hiding Table Headers .....	7-113
Using Custom Columns by Overriding createDefaultColumnsFromModel .....	7-113
Customizing the Table Body Generation by Overriding printRow .....	7-114
Customizing Header Generation by Overriding printHeaders .....	7-117
Access the contextObj and TemplateState in an HTML Component in the Table .....	7-117
Selecting a New Service Name for Custom Default Tag Attributes .....	7-118
Registering for the HTML Table Service .....	7-118
Using the HTML Table Service .....	7-119

Overview .....	7-119
BasicTableService .....	7-119
AssociationListTableService .....	7-121
ListContentTableService .....	7-123
Performing a Multiselect in the HTML Client .....	7-126
Generating a Column of Checkboxes .....	7-126
WTHtmlTable and Enabling the Checkbox .....	7-127
Multiselect Following a Generic Search .....	7-127
Multiselect on an Association Navigation .....	7-128
Multiselect on the Contents of a Business Object .....	7-130
Adding a Column of Checkboxes to Any Subclass of HTMLTable .....	7-132
Changing the Presentation of the Attributes in the HTML Client .....	7-133
Overview .....	7-133
How the Display is Generated by Default .....	7-134
Overriding the Display with an HTML Component .....	7-135
Adding a Link to an Attribute Value .....	7-136
Setting Color and Style Attributes in HTML and JSP Clients .....	7-142
Windchill Stylesheet .....	7-142
Color and Font Properties in wt.properties .....	7-143
HTML Component Implementation .....	7-146
The General Structure .....	7-146
The HTMLComponent Class .....	7-147
HTML Peer Components .....	7-149
Initializing Default Values .....	7-150
Custom Components .....	7-152
The Factory Service Implementation for HTML Components .....	7-152
HTML Table Generation and Table Service Implementation .....	7-154
HTMLTable .....	7-154
HTMLTableColumnModel .....	7-158
HTMLTableColumn .....	7-158
HTMLComponentFactory .....	7-159
HTMLValueObject and WTAttribute .....	7-160
WTHtmlTable .....	7-161
The Table Service .....	7-161
<b>Customizing PDMLink Template Processing Clients .....</b>	<b>8-1</b>
Directory Structure .....	8-2
Properties and Property Files .....	8-3
Template Framework .....	8-3
Template Processing Extensions for PDMLink .....	8-5

Action Configuration .....	8-5
Default ActionDelegates and URLActionDelegates .....	8-32
Tables.....	8-33
Common Customizations .....	8-46
How to Create a Soft-Type-Specific Actions Dropdown Menu .....	8-49
How to Add a Link to a Details Page Navigation Bar .....	8-53
How to Add a Soft-Type-Specific Details Page Navigation Bar .....	8-57
How to Add an Action to the Actions Column of a PDMLink Table .....	8-59
How to Add a Multiselect Action to the Multiselect Action Bar of a PDMLink Table.....	8-62
How to Add an Action to the Context-Specific Action Bar of a PDMLink Table .....	8-67
<b>Customizing Search Functionality .....</b>	<b>9-1</b>
Adding a Search Component.....	9-2
Introduction.....	9-2
Defining a Search Component .....	9-2
Using a Component.....	9-4
Overriding an Object's Life Cycle State Property.....	9-6
Searching for Soft Types and Attributes .....	9-7
Soft Types in Searches .....	9-7
Soft Attributes in Searches.....	9-7
The SearchableAttributes.properties File.....	9-8
Adding Actions to an Object in the Search Results Table .....	9-9
Adding a New Type to the Search User Interface.....	9-10
Basic Search Layout .....	9-10
Adding a New Type .....	9-12
<b>Customizing Change Management .....</b>	<b>10-1</b>
Overview .....	10-2
Prerequisite Knowledge .....	10-2
Change Objects and Their Relationships .....	10-3
HTML Hidden Input Form Fields .....	10-4
Expand and Collapse Functionality.....	10-4
Create and Update Functionality.....	10-5
Generating the Change Management User Interface .....	10-6
Template Processors .....	10-6
Form Task Delegates.....	10-16
Changing the Look of the User Interface .....	10-18
Changing Text.....	10-18
Changing Font.....	10-21
Changing Background Color .....	10-22
Removing or Changing Parts of the User Interface.....	10-24

Processor/Template Reference Tables .....	10-31
Action Links .....	10-31
Section Headers .....	10-33
Expandable/Collapseable Objects .....	10-33
Adding an Attribute to WTChangeRequest2 .....	10-34
Directory Structure .....	10-34
Work Overview .....	10-36
Rational Rose Tasks .....	10-37
SQL Tasks .....	10-38
Visual Café Tasks .....	10-38
Text Editor Tasks .....	10-41
Change Management Delegates .....	10-43
ChooseLifeCycleDelegate .....	10-43
ChooseFolderDelegate .....	10-43
ConcreteAssociationDelegate .....	10-44
DisplayIdentificationDelegate .....	10-45
<b>Customizing Foundation Applications .....</b>	<b>11-1</b>
Customizing Document Management .....	11-2
Customizing the HTML Client .....	11-2
Customizing Structure and References Processing .....	11-5
Customizing Create Document .....	11-7
Customizing Update Document .....	11-8
Customizing Check Out Document .....	11-9
Customizing Check In Document .....	11-10
Customizing Get Content .....	11-10
Customizing Create Document Template for a WTdocument Subtype .....	11-10
Customizing Create Document from Template from a WTdocument Subtype .....	11-12
Customizing Life Cycle .....	11-14
Customizing Workflow .....	11-16
Customizing Workflow HTML Templates .....	11-16
Customizing Change Management Workflow Process Templates .....	11-20
<b>Customizing Workgroup Managers .....</b>	<b>12-1</b>
Enabling Support for Custom Parts .....	12-2
Modifying File Types .....	12-2
Modifying Registry Files .....	12-3
Customizing Automatic Part Generation .....	12-5
Supporting the AutoPartGenerator Interface .....	12-5
Modifying the AutoPartCreator Preference .....	12-5

<b>Report Generation .....</b>	<b>13-1</b>
Overview .....	13-2
Basic Report Example .....	13-2
Query.....	13-2
Report Parameters .....	13-7
Import and Export of Report Templates.....	13-11
Customization Details .....	13-15
Customizing the Query.....	13-15
Customizing the Report Format .....	13-16
Customizing the Report Generation Client.....	13-30
Customizing the Report Generation URL.....	13-33
Customizing Macros.....	13-35
Customizing QueryBuilder Types.....	13-37
<b>Customization Examples .....</b>	<b>14-1</b>
Creating a Listener Service.....	14-2
Creating the projectaux Service .....	14-2
Testing the projectaux Service .....	14-5
Adding Events to the projectaux Service.....	14-6
<b>Xconfmanager Utility.....</b>	<b>15-1</b>
About the xconfmanager Utility .....	15-2
Formatting Property Value Guidelines .....	15-3
About the Windchill Command.....	15-5
About the windchill shell.....	15-7
<b>File-Handling Applets.....</b>	<b>16-1</b>
The Three Applets .....	16-2
Advantages and Disadvantages.....	16-2
The File Selection Applet .....	16-3
The Upload Applet.....	16-11
The Download Applet.....	16-31
<b>Enumerated Types.....</b>	<b>A-1</b>
The EnumeratedType Class .....	A-2
Creating an EnumeratedType Subclass .....	A-3
Editing the Resource Info for an Enumerated Type.....	A-7
Header.....	A-7
Resource Entry Format .....	A-7
Resource Entry Contents .....	A-8
Building Runtime Resources .....	A-8
Localizing an Enumerated Type .....	A-10

Extending an Enumerated Type .....	A-11
The Enumerated Type Customization Utility .....	A-12
Starting the Utility .....	A-13
GUI Usage of an Enumerated Type .....	A-13
<b>Extendable Classes in the Windchill Supported API .....</b>	<b>B-1</b>
PDM Business Information Classes .....	B-2
Enterprise Business Information Classes .....	B-2
Windchill Services .....	B-2
Foundation Classes .....	B-3
PDM Auxiliary Business Information Classes .....	B-3
Business Logic Classes .....	B-3
Server Development Classes .....	B-4
Client Development Classes .....	B-5



# Change Record

This chapter details major changes that have occurred since the last release.

**Table 1 Changes for Release 7.0**

Change	Description
Chapter 8, <a href="#">Customizing PDMLink Template Processing Clients</a>	This chapter was added for Windchill 7.0.
Chapter 9, <a href="#">Customizing Search Functionality</a>	This chapter was added for Windchill 7.0.
Chapter 15, <a href="#">Xconfmanager Utility</a>	This chapter was added for Windchill 7.0
Chapter 16, <a href="#">File-Handling Applets</a>	This chapter was added for Windchill 7.0



# About This Guide

The *Windchill Customizer's Guide* describes how to customize the out-of-the-box implementation of Windchill. It is intended for developers who are familiar with the information in the *Windchill Application Developer's Guide* and have the prerequisites necessary to use that book.

## Related Documentation

The following documentation may be helpful:

- *What's New for Windchill Release 7.0*
- *Windchill Installation and Configuration Guide*
- *Windchill Upgrade Guide*
- *Windchill Application Developer's Guide*
- *Windchill System Administrator's Guide*
- *Windchill Business Administrator's Guide*
- *Windchill Performance Tuning Guide*
- *Windchill User's Guide*

If books are not installed on your system, see your system administrator.

## Technical Support

Contact PTC Technical Support through the PTC Web site, phone, fax, or e-mail if you encounter problems using Windchill.

For complete details, refer to Contacting Technical Support in the *PTC Customer Service Guide* enclosed with your shipment. This guide can also be found under the Support Bulletins section of the PTC Web site at:

<http://www.ptc.com/support/index.htm>

The PTC Web site also provides a search facility that allows you to locate Technical Support technical documentation of particular interest. To access this page, use the following link:

<http://www.ptc.com/support/support.htm>

You must have a Service Contract Number (SCN) before you can receive technical support. If you do not have an SCN, contact PTC License Management using the instructions found in your *PTC Customer Service Guide* under Contacting License Management.

## Documentation for PTC Products

PTC provides documentation in the following forms:

- Help topics
- PDF books

To view and print PDF books, you must have the Adobe Acrobat Reader installed.

All Windchill documentation is included on the CD for the application. In addition, books updated after release (for example, to support a hardware platform certification) are available from the Reference Documents section of the PTC Web site at the following URL:

<http://www.ptc.com/cs/doc/reference/>

## Comments

PTC welcomes your suggestions and comments on its documentation -- send comments to the following address:

[documentation@ptc.com](mailto:documentation@ptc.com)

Please include the name of the application and its release number with your comments. For online books, provide the book title.

## Documentation Conventions

Windchill documentation uses the following conventions:

Convention	Item	Example
<b>Bold</b>	Names of elements in the user interface such as buttons, menu paths, and dialog box titles.  Required elements and keywords or characters in syntax formats.	Click <b>OK</b> . Select <b>File &gt; Save</b> . <b>License File</b> dialog box <b>create_&lt;tablename&gt;.sql</b>
<i>Italic</i>	Variable and user-defined elements in syntax formats. Angle brackets (< and >) enclose individual elements.	<b>create_&lt;tablename&gt;.sql</b>
Monospace	Examples  Messages	JavaGen "wt.doc.*" F true Processing completed.
"Quotation marks"	Strings	The string "UsrSCM" . . .



# 1

## Customizing Modeled Business Objects

<b>Topic</b>	<b>Page</b>
Windchill Customization Points.....	1-2
Handling Icons .....	1-3
Creating Large Objects (LOBs) .....	1-5

# Windchill Customization Points

Windchill is composed of thousands of Java classes. To help you focus your customization efforts, the Windchill Java classes have been partitioned into two sets. To customize Windchill, you should interact with the classes in the Windchill Supported API set.

## Windchill Supported API

The Windchill Supported API includes those classes that customizers are meant to work with directly. A class might be in the Supported API because it is meant to be extended by customizers or, more likely, because it has some methods for customizers to call. Programming elements that are part of the Supported API will not be changed without notification and a deprecation period, whenever possible. Deprecation periods will extend up to two releases after notification and the Javadoc for a deprecated element will state the release at which the element will no longer be available.

Classes, methods, and other programming elements that are not part of the Supported API should not be used directly by customizers. Those elements are subject to change without notification or a deprecation period.

To determine if a class, or one of its methods, is part of the Supported API, consult the Javadoc for that class. The Javadoc for a class contains one or two lines that indicate if the class is part of the Supported API. For example, the `WTPart` class is part of the Supported API, and, in addition, it is extendable; therefore, it has two lines that indicate this (as shown in the following illustration).

### Class `wt.part.WTPart`

```
java.lang.Object
|
+----wt.fc.WTObject
      |
      +----wt.enterprise.RevisionControlled
            |
            +----wt.part.WTPart
```

---

```
public class WTPart
extends RevisionControlled
implements PartVersion, ContentHolder, EffectivityManageable, ViewManageable, Baselineable,
Externalizable
```

The reference implementation of a `PartVersion`. It can be checked in and out, assigned to views, baselined, assigned to an effectivity, and can also hold content.

Use the `newWTPart` static factory method(s), not the `WTPart` constructor, to construct instances of this class. Instances must be constructed using the static factory(s), in order to ensure proper initialization of the instance.

**Supported API:** true

**Extendable:** true

The fact that a class is part of the Supported API indicates that some part of it is meant to be used by customizers. A class is meant to be extended only if its Javadoc contains the line indicating "Extendable: true". (Classes that can be extended are listed in Appendix B, Extendable Classes in the Windchill Supported API.) If the Javadoc states "Support API: false", then the class and its methods are not part of the Support API.

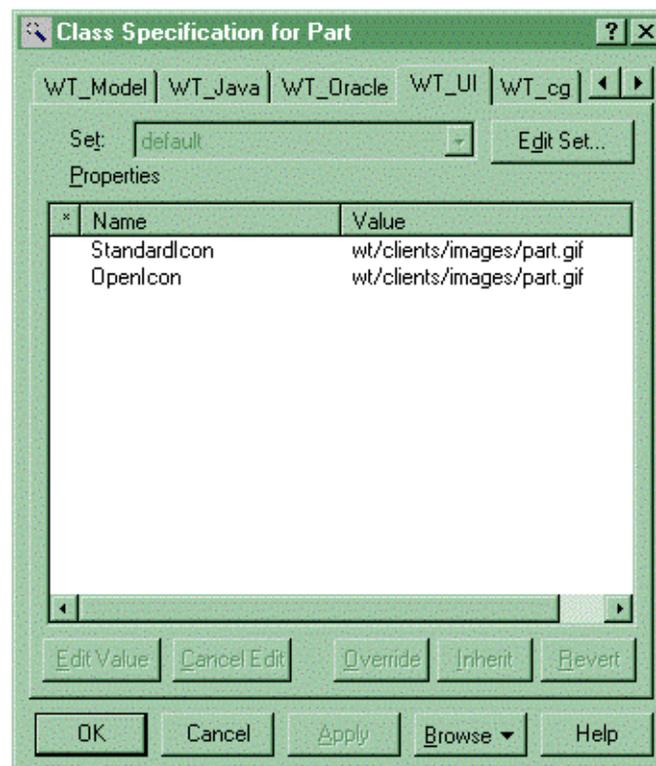
Methods, and other programming elements, also have a "Supported API" line in their Javadoc. If a class is not part of the Supported API, then none of its methods are either. If a class is part of the Supported API, that does not indicate that its methods are, too. For a method to be part of the Supported API, its Javadoc must also state "Supported API: true".

## Javadoc

Javadoc is distributed for some, but not all, Windchill classes. Of those classes for which Javadoc is distributed, some, but not all, are part of the Supported API. Javadoc is distributed for some classes for informational purposes, even though they may not be part of the Supported API.

## Handling Icons

The icons for a class are modeled into Rose. For the modeled class, select the WT\_UI tab and set the StandardIcon and OpenIcon accordingly.



These paths must be fully qualified from the Windchill codebase. The location for all gif images at this time is wt/clients/image.

The gif image specified must be 16 by 16 pixels.

Class inheritance determines what glyphs are overlaid on an object:

- Classes that are subclass RevisionControlled are overlaid with a WORKING or a CHECKED-OUT glyph when appropriate.
- Subclasses of wt.doc.Document are displayed based on their primary format and will also inherit the overlays of RevisionControlled.
- Classes that implement the wt.content.FormatContentHolder interface directly are displayed based on the primary format.
- Shortcuts are displayed based on the target of the shortcut.

wt.clients.util.IconCache is the class that clients use to access icons. An example of its usage follows:

```
IconCache icons = new IconCache( someWTContextObj );  
Image img = icons.getStandardIcon( someWTObject );  
Image img2 = icons.getOpenIcon( someWTObject );
```

Any subclass of wt.fc.WTObject is accepted as an argument. See the Javadoc on this class for more information.

# Creating Large Objects (LOBs)

## Modeling Large Objects

You can model an attribute to be a binary large object (BLOB) in one of the following four ways:

- By default when the attribute type is a class that does not map to any of the other `java.sql.Types`.
- Explicitly set the attribute's class to `LobLocator`. Use this technique for large objects that you access infrequently.
- Model the attribute to map to a `SMALLBLOB` by changing the `ColumnType` property under the Oracle tab of the attribute specification. Use this technique for objects that can expand and contract, but are never more than a certain size.
- Model the attribute to map to an `INLINEBLOB` by changing the `ColumnType` property under the Oracle tab of the attribute specification. Use this technique for objects that can expand and contract, and are typically small in size, but occasionally require large storage size (such as, adhoc ACLs).

## DDL Generation

The DDL generated by Windchill defines the BLOB columns to be stored in their own tablespace. This can be controlled for each attribute by changing the tablespace property under the Oracle tab while modeling. The tablespace must be defined before the DDL is executed. The name of the default tablespace to contain BLOBs can be modified using a property in the `user.properties` file. See the `properties.html` file for details on the specific property.

## Reading and Writing LOBs

BLOBs are either read or written as byte arrays or as Serialized objects each time the object containing the BLOB is read, created, or updated. If the BLOB is modeled using `LobLocator`, then only the LOB locator is retrieved with each read/write and a second step is required to retrieve or store the LOB. The `StandardPersistenceManager` provides `getLob()` and `updateLob()` to perform these operations.

Be careful about the types of objects that you serialize into the database. Using default Java serialization is especially prone to breaking if the Java class files are recompiled with possible changes, or sometimes just with different compilers. Programming to allow upward compatibility of serialized representations is difficult. One technique is as follows:

1. Make the class `Externalizable`, which aids performance as well.
2. Specify a fixed `serialVersionUID` value.

3. Put an internal version number on the object stream and handle version changes in code.

See the code-generated Externalizable business classes for an example of this technique. The disadvantage is that all subclasses are now Externalizable and require readExternal and writeExternal methods. You can use the same type of technique with Serializable (fix serialVersionUID and implement read/writeObject) if independence from the class hierarchy is important.

## Small BLOBs

SMALLBLOBs are encoded into Strings before being stored and are mapped to VARCHAR2(4000) rather than BLOB. Because of the size limitation of 4,000 characters, this option is inflexible and should be used with caution. Your code should be prepared to handle the possible exception that is thrown if the size of the attribute grows beyond what can be encoded into 4,000 characters.

## Inline BLOBs

INLINEBLOBs combine the features of SMALLBLOB and BLOB to provide better performance when data is small, but still allow for large data storage capability. An attribute that is stored as an INLINEBLOB uses two columns:

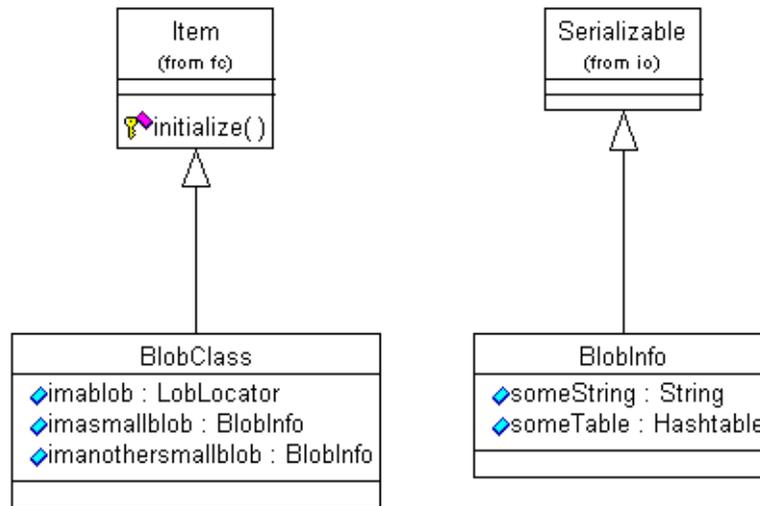
- A VARCHAR2(4000) column
- A BLOB column

Each object is examined at runtime and, when the data is small enough, it is stored in the VARCHAR(4000) column; otherwise, it is stored in the BLOB column. When the data is stored in the VARCHAR(4000) column, no additional datastore calls are required to read the BLOB data.

Using an INLINEBLOB introduces a fixed cost in terms of storage (two columns are used instead of one) and additional processing at runtime. However, when the majority of data is stored “inline” with the table, this cost is negligible in comparison to the performance benefits. If most of the data exceeds the 4000-byte limit, BLOB storage is probably more appropriate. However, if it is known that all data will never exceed the 4000-byte limit, SMALLBLOB should be used.

## Example

An example follows, which uses the classes shown in the following figure.



```
/*
 *
 * Lobber is an example of storing BLOBs. BlobClass has three BLOBs:
 * imablob which is LobLocator, and imasmallblob, and imanothersmallblob
 * which are
 *
 */
```

```
public class Lobber {

    private static final int size = 1000;
    private static BlobClass atr = null;
    private static Enumeration lob_files;

    public static void goLobs( String lob_dir ) {

        try {

            // create a BlobClass object

            lob_files = getLobFiles( lob_dir );

            // loop thru the files in this directory
            if ( lob_files != null ) {
                if ( lob_files.hasMoreElements() ) {
                    try {
                        String lobIt = (String)lob_files.nextElement();
                        File lobFile = new File(lobIt);

                        long len = lobFile.length();
                        System.out.println( "file length "+len );
                        InputStream aIs = new FileInputStream(lobFile);
```



```
    }  
}  
  
private static Enumeration getLobFiles( String ldir ) {  
  
    String[] s_files = null;  
    Vector aVec = new Vector();  
    File lob_dir = new File( ldir );  
  
    if ( !lob_dir.isDirectory() )  
        return null;  
  
    s_files = lob_dir.list();  
    for ( int i=0; i  
    for ( int i=0; i
```



# 2

## Customizing Modeled Elements

Topic	Page
Customizing Column Lengths.....	2-2

# Customizing Column Lengths

A column length for a modeled attribute can be customized. These column lengths are obtained through the `wt.introspection` package. The default column lengths are defined in the delivered Rose models with the attribute's `UpperLimit` property. The value of this property can be overridden by placing entries in the customizations property file for modeled packages.

To change the column length for a modeled attribute, perform the following steps:

1. Determine which customization property entry must be added.
2. Add the customization entry to the appropriate customizations property file.
3. Generate the class info objects and sql scripts.
4. Verify the customization.
5. Create the database tables.
6. Restart the method servers if they were running during the procedure.

The following example sets the column length for the **name** attribute of the `wt.doc.WTDocumentMaster` class to 350. The following steps describe how to determine which customizations property file entry will contain the new column length, and how to set the value. The customization will be made in a location parallel with the originally modeled attribute.

The default location for these customizations is `$(wt.home)\wtCustom`, as defined by the `wt.generation.custom.dir` entry in `tools.properties`. Create this directory if it does not already exist.

1. Determine which customization property entry must be added:

- a. Obtain an info report for the class by executing the following command:

```
infoReport wt.doc.WTDocumentMaster
```

- b. Inspect the value of the `WTIntrospector.UPPER_LIMIT` property (the value being customized) of the **name** PropertyDescriptor:

```
getValue( WTIntrospector.UPPER_LIMIT ) : 60
```

- c. Inspect the value of the `WTIntrospector.DEFINED_AS` property of the **name** PropertyDescriptor:

```
getValue( WTIntrospector.DEFINED_AS ) :  
wt.doc.WTDocumentMaster.name
```

- d. Based on this information, use the following values:

- The customization property file is `Windchill\wtCustom\wt\doc\docModel.properties`.
- The customization property entry is `"WTDocumentMaster.name.UpperLimit"`.

2. Add the customization property entry to the appropriate customizations property file. In this example, add the following entry to Windchill\wtCustom\wt\doc\docModel.properties (create this file if it does not exist):

```
WTDocumentMaster.name.UpperLimit=350
# ignore multi-byte database character sets when setting value
```

3. Generate the class info objects and the sql scripts:

- a. Update the serialized info object and the sql script for the customized class by entering the following command (on one line):

```
%wt_home%\bin\tools.xml custom_column -Dgen.input=wt.doc.*
```

- b. Inspect the infoReport's descendents list for the WTDocumentMaster class:

```
getDescendentInfos():

    ClassInfo wt.build.buildtest.SourceMaster
    ClassInfo wt.federation.ProxyDocumentMaster
```

- c. If the customized class has descendants that are concrete, their tables must also be adjusted with the following commands (one line each):

```
ant -f %wt_home%\bin\tools.xml custom_column -
Dgen.input="wt.build.buildtest.SourceMaster

ant -f %wt_home%\bin\tools.xml custom_column
-Dgen.input=wt.federation.ProxyDocumentMaster
```

4. Verify the customization:

- a. Obtain an info report for the class and inspect the UPPER\_LIMIT value as described in the preceding steps. The value should reflect the customization.

- b. If the info report value is unchanged, perform the following steps:

- i. Verify that the generate step actually updated the following serialized info file:

```
Windchill\codebase\wt\doc\WTDocumentMaster.ClassInfo.ser
```

- ii. Verify that a class or interface in WTPartMaster's hierarchy does not also define the **name** attribute. This is done most easily by viewing the **Attributes** tab of the **Class Specification** dialog in Rose, with **Show inherited** selected. If the name attribute is defined in the hierarchy, the **name** attribute that was modeled originally must be customized.

Create the database tables. If the tables already exist, adjust the length of the customized column.

Because, in this example, `WTDocumentMaster.name` is also the source for the derived attribute `WTDocument.name`, the derived attribute must also be customized in this manner. This customization is necessary because the derived attribute, `WTDocument.name`, does not get its `UpperLimit` from the source attribute, `WTDocumentMaster.name`. The derived attribute also explicitly set the `UpperLimit` property in the model. Therefore, the customization must be defined explicitly for the derived attribute `WTDocument.name`. Following the preceding steps would determine that a `WTDocument.name.UpperLimit` entry would be set in the customization property file `Windchill\wtCustom\wt\doc\docModel.properties`. This additional step applies specifically to the way derived attributes work and is not necessary for normal attributes.

# 3

## Customizing Business Logic

<b>Topic</b>	<b>Page</b>
Identified Business Classes .....	3-2
Customizing Iteration and Version Identifiers .....	3-17
Customizing Indexing Behavior.....	3-19
Customizing a Bill of Materials .....	3-28
Customizing User Preferences .....	3-32
Writing a Servlet Helper to Use URLFactory Functionality.....	3-36

# Identified Business Classes

## Identity Attributes

Identity attributes are the attributes of an object that, as a whole, distinguish it to the user from other objects. The designer of a class must decide which attributes compose its identity. For example, a SubFolder is identified by its name and its parent folder. A WTPart's identity attributes are number and name. At least one of the identity attributes of a class must have a uniqueness constraint in order to distinguish between different instances of the class.

## How Identity Is Assigned and Managed

### User-Assigned Identity Attributes

User-assigned identity attributes can be set by a user when an object is created. Unlike other attributes, they cannot be modified after the object is created. However, the user can change the values of identity attributes using the rename operation in Windchill Explorer. This is actually accomplished by calling the changeIdentity operation of the IdentityService. The name attribute of the WTDocument class is an example of a user-assigned identity attribute.

### System-Assigned Identity Attributes

System-assigned identity attributes have values that are generated by the system when the object is created. The number attribute for the WTChangeOrder2 class is an example of a system-assigned identity attribute.

### Changing an Object's Identity

Currently, all Windchill classes that have user-assigned identity attributes can be modified later by the user, but system-assigned identity attributes cannot be modified by the user. This is not a hard and fast rule, however. For example, a document number could be assigned by a user when the document is created and then never be allowed to change. Or, a default number could be generated by the system, but the user could override this number if desired either when the document is created or at a later date.

## User-Managed Versus System-Managed Identity

An object's identity is user-managed if the user can either assign or modify its identity attributes. It is important to distinguish user-managed from system-managed identity because typically not all identity changes that a user might attempt are valid. For example, a part master is identified by its name and number. `WTPartMaster` is user-managed because the user can assign and modify a part master's name and number. Because it is user-managed, a constraint is enforced which requires that no two parts can have the same number. If the part number was assigned by the system, however, it would no longer be necessary to enforce the part number uniqueness constraint because the algorithm that the system uses to generate part numbers guarantees unique part numbers.

The interfaces `wt.fc.Identified` and `wt.fc.UniquelyIdentified`, which work in conjunction with the `IdentityService`, are implemented by classes that have user-managed identity in order to apply validation logic and uniqueness constraints when a user makes a change to an object's identity.

## A Special Case—RevisionControlled Identity

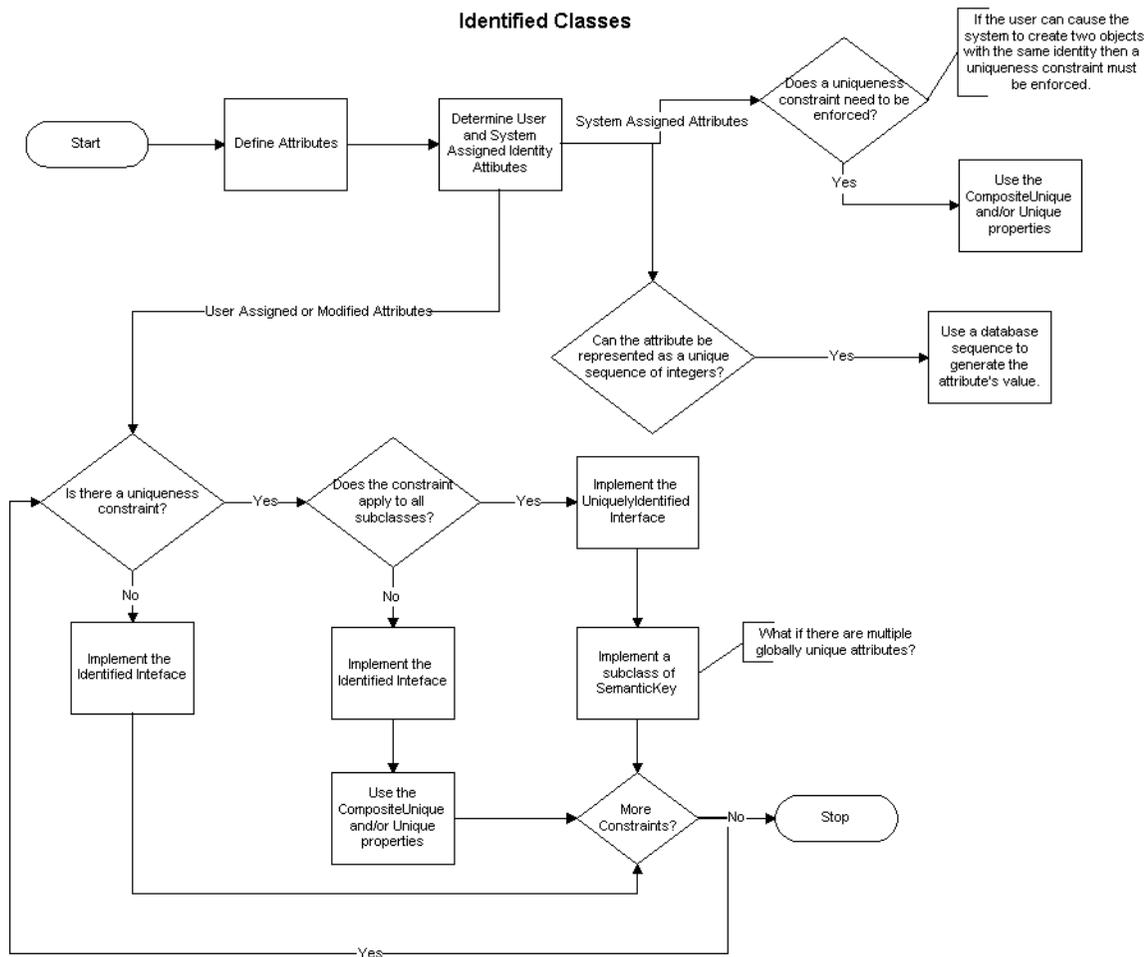
Classes that extend `RevisionControlled` are identified by a version and iteration identifier. The version and iteration are assigned by the system, based on the previous version and iteration. However, because a new version is derived from a user-specified old version, possibly not the latest, a user may cause the same system-generated identity to apply to two separate objects. Because of this behavior, `RevisionControlled` does not fit the description of either a user- or system-managed identity. A uniqueness constraint must be enforced such that two identical versions of the same master cannot be created. However, subclasses of `RevisionControlled` do not need to implement the `Identified` interface, as explained next.

Checking out a `RevisionControlled` object, such as a `WTDocument`, causes the document's iteration number to increase: A.1 to A.2. In this case, it is clear that the system is managing the identity, and it is impossible for the user to create two different iterations with the same number. Creating a new version conceptually creates a new document with an identity of B.1. Again the system is managing the version identity attribute. However, when both version A.2 and B.1 exist, it is possible for a user to attempt to create a new version from A.2 again. If not prevented, this would create two version B objects. Thus, a uniqueness constraint must be applied that prevents two identical versions of the same master from being created. This situation is unique because the system manages the identity attributes, but the user can instruct the system to create the same object twice, even though it does not directly control the assignment of the version letter. Contrast this to a class such as `WTChangeRequest2`, where each new change request object is assigned the next unique number from an integer sequence for its identity.

## When to Use the IdentityService

Any class that has user-managed identity attributes must implement the Identified or UniquelyIdentified interfaces. This enables the object's identity to be changed with the IdentityService. The IdentityService provides users of a class that implements the Identified interface a means to change the identity attribute values subject to validation logic and uniqueness constraints.

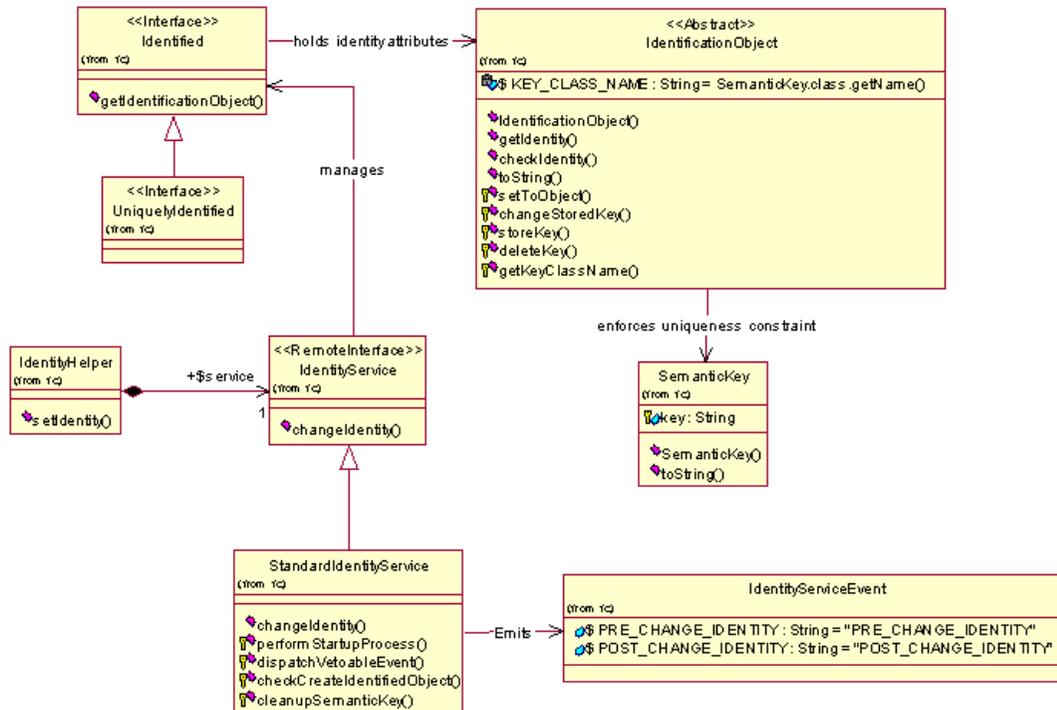
Use the following flowchart to help determine whether a class that has identity attributes should implement the Identified interface, the UniquelyIdentified interface, or neither.



## How to Use the IdentityService

### Overview

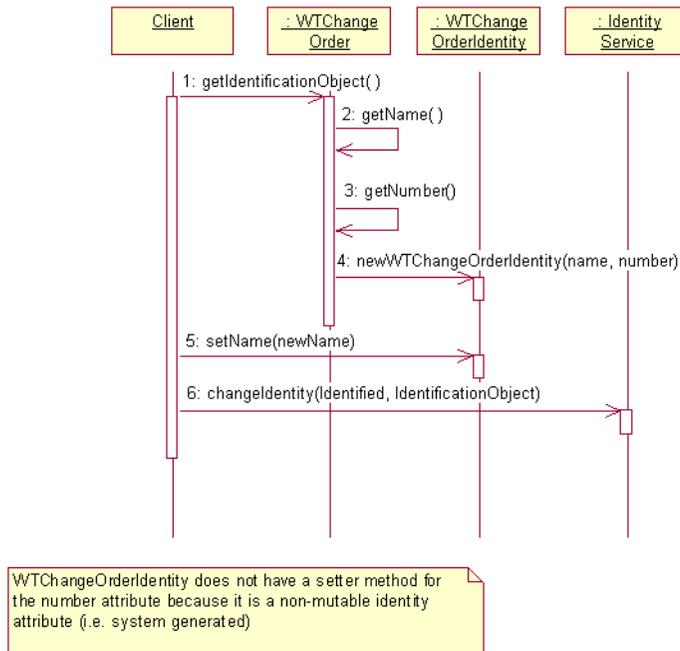
The following figure illustrates the IdentityService.



IdentificationObject is a non-persistent class that holds the values of the identity attributes from a class that implements the Identified or UniquelyIdentified interface. For a particular business class that implements Identified, a subclass of IdentificationObject is modeled that has the same identity attributes as the business class. Instead of directly changing the identity attributes of the business object, the attributes of the IdentificationObject subclass are used to change the identity, and these values are then copied back to the business object, subject to validation logic and uniqueness constraints, when IdentityService.changeIdentity is called.

Any class that implements the Identified interface must implement the getIdentificationObject method, which returns an instance of IdentificationObject that holds the current value of the identity attributes.

The following sequence diagram illustrates how a change order's identity is modified:



## Identified

When a class implements the Identified or UniquelyIdentified interface, the identity attributes of the class must be modeled in a special way. Also, a single method on the Identified interface -- `getIdentificationObject` -- must be implemented. UniquelyIdentified has no methods to implement.

`wt.change2.WTChangeRequest2` is used to illustrate the step-by-step instructions. See the class diagram named "request" in the `wt.change2` package of the Rose model for reference.

1. Add a public constructor that has arguments for each user-assigned identity attribute.

`WTChangeRequest2` has a constructor that has a single argument for the name attribute. Number is the other identity attribute but, because it is system-assigned, it does not need to be an argument of the constructor. In the `initialize` method that is called from the generated factory method for this constructor, the name attribute is initialized. Sequential system-assigned attributes, like numbers, should not be initialized in the constructor, but instead in the `PersistenceManager PRE_STORE` event processing right before the object is saved. This greatly improves the odds that elements in the sequence are not lost if the create action is aborted by the user. That is, system-assigned attributes are initialized only when it is certain that an object will be persisted.

2. Set the WriteAccess model property to "protected" for all identity attributes.

This action prevents general access to the setter methods, allowing only the IdentityService to update the identity attributes of an Identified object.

**Note:** There may be cases where it is necessary to create an uninitialized instance of a class by calling the no-arg constructor, followed by calls to the setter methods to initialize the attributes. This is the case with WTPartMaster and WTDocumentMaster. In this situation, the identity attributes must have public WriteAccess instead of protected. To prevent the use of the setter methods after the object has been persisted, use the "Changeable" property. When set to the value "ViaOtherMeans", the Changeable property causes validation code to be generated in the setter method for the attribute, which allows the method to be used only if the object is not persistent.

3. Implement getIdentificationObject (part of the Identified interface) to return a new instance of the subclass of IdentificationObject that has been designed for this particular Identified class. In the case of WTChangeRequest2, there is a subclass of IdentificationObject called WTChangeRequest2Identity. Following is an example:

```
public IdentificationObject getIdentificationObject()
    throws WTEException {
    ///begin getIdentificationObject%3536588B01EEg.body
    preserve=yes
    return WTChangeRequest2Identity.newWTChangeRequest2Identity(
        getNumber(), getName());
    ///end getIdentificationObject%3536588B01EEg.body
}
```

The following section describes the role of IdentificationObject and how to extend it.

## IdentificationObject

Throughout these instructions, the example of wt.folder.SubFolderLinkIdentity is used to illustrate each step. See the class diagram named "Identification" in the wt.folder package of the Rose model for reference.

1. Model a subclass of IdentificationObject in the same package.

wt.folder.SubFolderLinkIdentity extends IdentificationObject. Notice that the subclass of IdentificationObject is in the same package as the class that implements Identified, namely SubFolderLink. This is a requirement because SubFolderLinkIdentity must call protected setter methods on SubFolderLink to change the value of its identity attributes.

2. Add public attributes for each user-assigned identity attribute.

For every user-assigned identity attribute of the Identified class, add the same attributes to the subclass of IdentificationObject. In the case of SubFolderLinkIdentity, the identity attributes are subFolderName and parentFolderReference. Making the identity attributes public on the IdentificationObject allows any client to assign a new identity to an object through the IdentificationObject setter methods instead of the ones on the Identified object (see the preceding figure on modifying a WTChangeOrder2's identity).

3. Add a protected constructor that has an argument for the Identified object.

The constructor is protected because it is intended to be invoked only in the getIdentificationObject method of the corresponding Identified class, in this example, SubFolderLink. Alternatively, the constructor can have an argument for each identity attribute instead of the entire Identified object.

4. Implement the code-generated initialize method to set the identity attributes from the Identified Object.

For every modeled constructor, a static factory method is automatically generated that is implemented to call an initialize method. In this initialize method, call the identity attribute getter methods on the Identified object and set them to the IdentificationObject's identity attributes. That is, copy the identity attributes from the Identified object to the IdentificationObject. Following is an example:

```
protected void initialize( SubFolderLink link )
    throws WTEException {
    ///##begin initialize%356993D30199.body preserve=yes

    setParentFolderReference((ObjectReference)link.
        getRoleAObjectRef());
    setSubFolderName(link.getFoldered().getName());

    ///##end initialize%356993D30199.body
}
```

5. Implement the getIdentity method.

The getIdentity method is not used unless the subclass of IdentificationObject is for a UniquelyIdentified class. If your class implements only Identified, getIdentity can return any String representation of the identity attributes.

6. Implement the setToObject method.

The setToObject method is invoked by the IdentityService to transfer the values of the identity attributes from the IdentificationObject to the Identified object. Following is the implementation of SubFolderLinkIdentity.setToObject:

```
public void setToObject( Identified obj ) {
    ///begin setToObject%356993A900B2s.body preserve=yes
    // All we have to set is the reference because the name
    // is managed by the SubFolder object.

    ((SubFolderLink)obj).
        setRoleAObjectRef(getParentFolderReference());

    ///end setToObject%356993A900B2s.body
}
```

This completes the implementation of the IdentificationObject subclass if the corresponding business class implements Identified. If it implements UniquelyIdentified, then the following two additional steps are necessary:

1. Override the getKeyClassName method.

This method returns the name of the class that extends SemanticKey, which is used to store the key value that is subject to the uniqueness constraint for the UniquelyIdentified object. For the SubFolderLinkIdentity example, a private, static String constant is modeled and initialized to the value SubFolderLinkConstraint.class.getName(). Then getKeyClassName simply returns this String constant.

2. Ensure that getIdentity returns a String that represents the uniqueness constraint on the UniquelyIdentified object.

Because getIdentity is an abstract method in IdentificationObject, every subclass must implement it. However, it is not actually used unless the subclass of IdentificationObject is for a UniquelyIdentified class. In this case, getIdentity is called to set the value of SemanticKey.key. Remember, this is the String that represents the uniqueness constraint on the UniquelyIdentified object. In this example, getIdentity returns the following string:

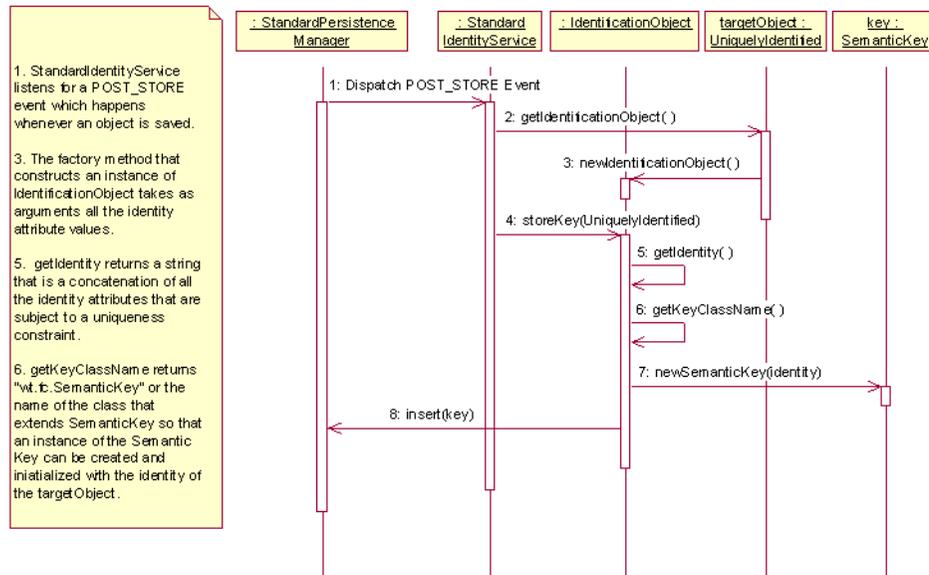
```
getParentFolderReference().getObjectId().getStringValue()
+ "+" + getSubFolderName();
```

This String is a combination of the parent folder and subfolder name. This indicates the uniqueness constraint on SubFolderLink is that no two subfolders in a given parent folder can have the same name.

## UniquelyIdentified and SemanticKeys

You must implement the UniquelyIdentified interface in cases where a uniqueness constraint on a user-managed identity attribute applies across multiple classes. This is because every class in the model is mapped to a single database table. Because unique database indexes apply to one table at most, some mechanism in the application layer must be used to enforce the uniqueness constraint. This mechanism is the SemanticKey class found in the wt.fc package.

The following sequence diagram illustrates how SemanticKey works:



The preceding figure illustrates the general case. Consider a specific example, `WTPartMaster`. `WTPartMaster` has two identity attributes: name and number. Number must be unique across all subclasses of `WTPartMaster`, so we must implement a `SemanticKey`. `WTPartMasterKey` extends `SemanticKey` and `WTPartMasterIdentity` extends `IdentificationObject`. The implementation of `WTPartMasterIdentity.getKeyClassName` returns "wt.part.WTPartMasterKey" and `WTPartMasterIdentity.getIdentity` returns the part number. So, whenever a `WTPartMaster` is saved, the value of the part number is stored in the `WTPartMasterKey` table. Any subclass of `WTPartMaster` also stores its number in the same `SemanticKey` table, `WTPartMasterKey`. Because there is a uniqueness constraint on the key attribute of the `WTPartMasterKey` class, unique part numbers are guaranteed across `WTPartMaster` and all its subclasses.

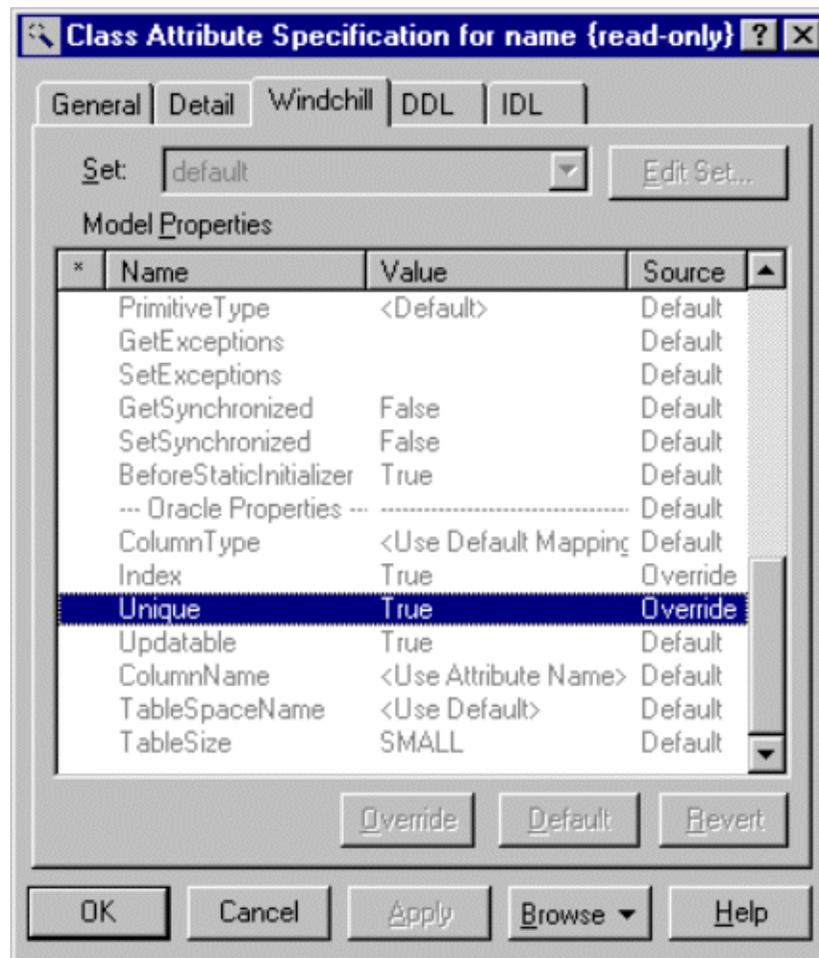
## Enforcing Uniqueness Constraints with Database Indexes

Windchill supports the creation of unique database indexes through the model properties Unique and CompositeUnique. Unique database indexes are an efficient way to enforce uniqueness constraints for one or more attributes of a single class. If the uniqueness constraint must apply across multiple classes, such as a base class and all its subclasses, database indexes do not help. In this case, SemanticKey class is used.

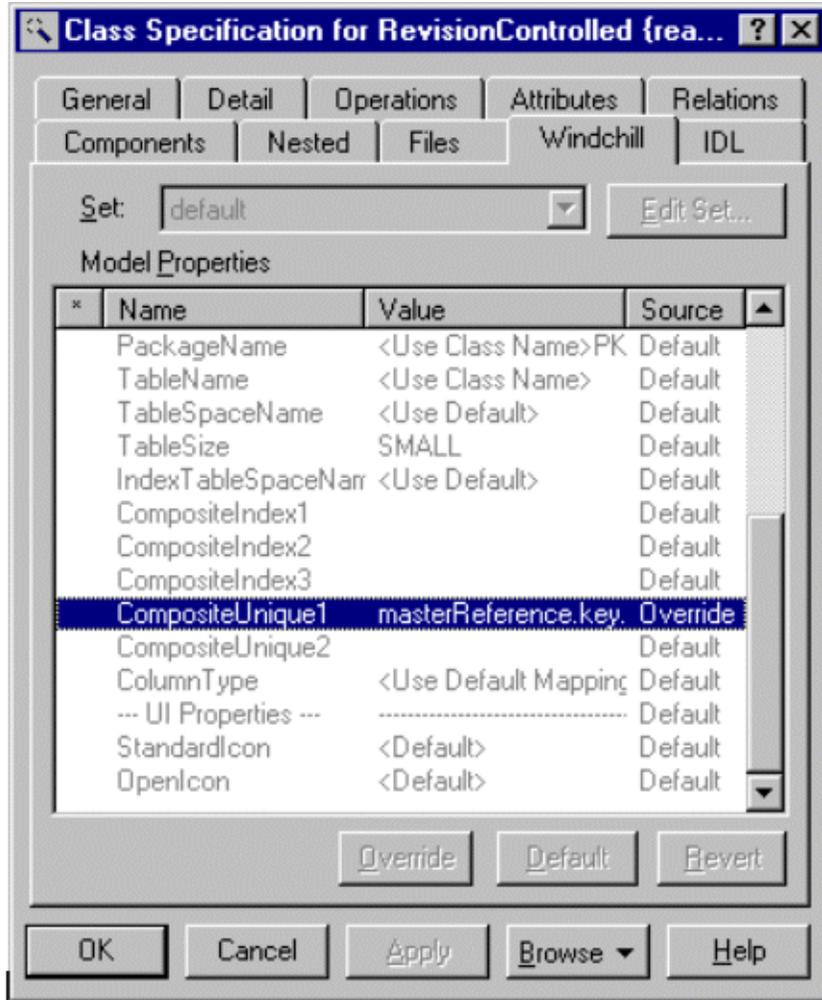
Database indexes are used to enforce uniqueness constraints on both user- and system-managed identity attributes. See the first figure in this chapter, the identified classes flowchart, to help determine when a database index should be used.

The Unique property of an attribute creates a unique index on the database column for that attribute. The following example is for wt.folder.Cabinet, which specifies that its name is a Unique attribute. The SQL DDL generated is as follows:

```
CREATE UNIQUE INDEX Cabinet$name ON Cabinet(name) STORAGE  
( INITIAL 20k NEXT 20k PCTINCREASE 0 )
```



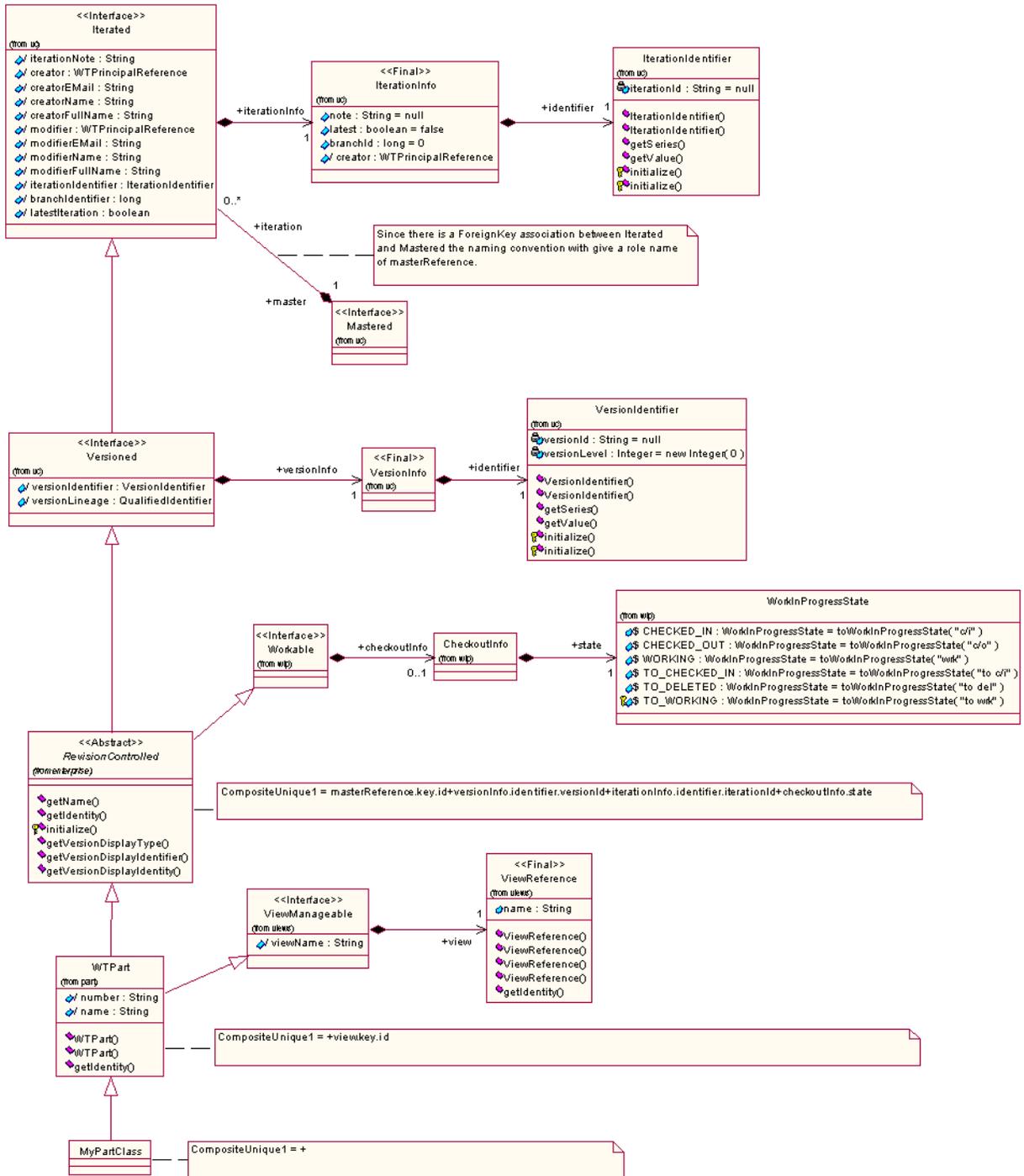
The CompositeUnique1 and CompositeUnique2 properties of a class create a unique index across multiple database columns. The following example is for wt.enterprise.RevisionControlled, which specifies that the master + version + iteration + checkout state attributes are unique as a whole.



When using the CompositeUnique property, only two unique indexes can currently be specified in the model. Additional attributes can be added to the index in the subclass by adding them after the "+".

The following figure is an example of a composite uniqueness constraint for WTPart.

Uniqueness constraint for a WTPart is enforced with the CompositeUnique model property. The constraint is that the combination of View, Master, Version, Iteration, and CheckOut State must be unique across all parts.



## How to Implement System-Managed Identity

To implement system-managed identity, perform the following steps:

1. Use database sequences for system-assigned number attributes.

Typically, system-assigned attributes are generated as a number sequence. Windchill supplies a utility method that provides access to a sequence of number values maintained by the Oracle database:

```
public static String
PersistenceHelper.manager.getNextSequence(String sequenceName);
```

Given a sequence name known to Oracle, the `getNextSequence` method returns a number that is represented as a `String`. The number value returned is incremented each time the `getNextSequence` method is invoked for the given sequence name.

2. Listen for the `PRE_STORE` event on the identified object.

The service associated with the Identified object in question must listen for a `PersistenceManager PRE_STORE` event and assign the number value at this time. For `WTChangeRequest2`, the `ChangeService2` implements the following to assign numbers to change requests:

```
protected void performStartupProcess()
    throws ManagerException {
    //##begin performStartupProcess%35D1B483001F.body preserve=yes
    /*
    * Listen for a PersistenceManager PRE_STORE event
    * on a WTChangeRequest2
    */
    getManagerService().addEventListener(
        new ServiceEventListenerAdapter() {
            public void notifyVetoableEvent( Object event )
                throws WTEException {
                PersistenceManagerEvent pmEvent =
                    (PersistenceManagerEvent)event;
                Persistable target = pmEvent.getTarget();
                if (target instanceof WTChangeRequest2) {
                    // assign a number
                    WTChangeRequest2Identity idObj =
                        (WTChangeRequest2Identity)target.
                            getIdentificationObject();
                    idObj.assignToObject(target);
                }
            }
        },
        PersistenceManagerEvent.generateEventKey(
            PersistenceManagerEvent.PRE_STORE));
}
```

- Implement the `assignToObject` method on `WTChangeRequest2Identity`.  
Following is `WTChangeRequest2Identity.assignToObject`:

```
protected void assignToObject( Identified obj )
    throws WTEException {
    ///##begin assignToObject%356DC85D038A.body preserve=yes

    String number = PersistenceHelper.manager.getNextSequence(
        "WTCHANGEREQUESTID_seq" );

    ((WTChangeRequest2)obj).setNumber(number);

    ///##end assignToObject%356DC85D038A.body
}
```

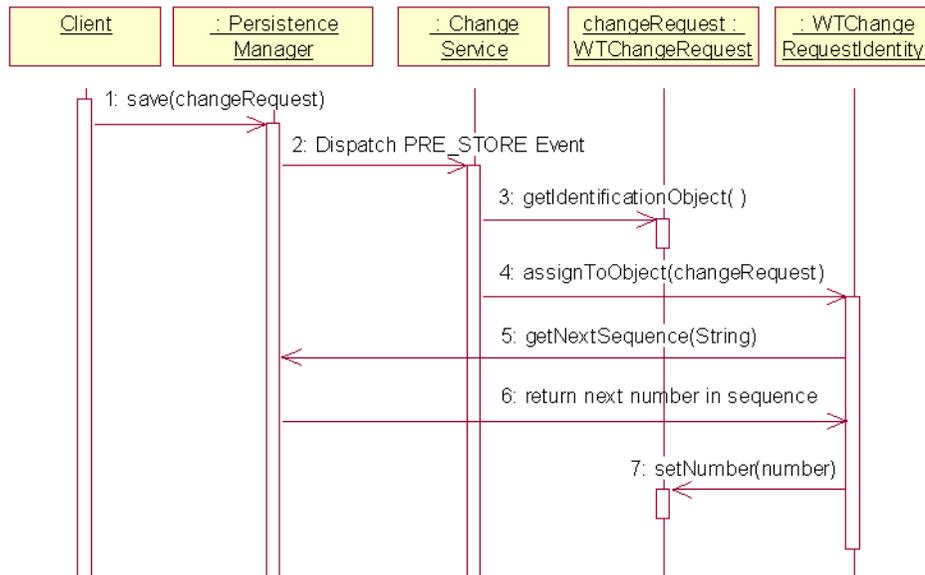
- Modify `WTChangeRequest2_UserAdditions.sql` using the following command:

```
exec wtpk.createSequence('WTCHANGEREQUESTID_seq',1,1)
```

In general, the syntax for creating a sequence is as follows:

```
wtpk.createSequence('<sequence name>', <starting number>,
    <increment> )
```

Following is an example for `WTChangeRequest2`:



## Customizing Iteration and Version Identifiers

Both iteration and version series can be customized to better represent a company's way of identifying minor and major changes, respectively. The following is a segment of the wt.properties file where properties exist for these kinds of customizations:

```
# Seriesfactoryassociations'  
wt.vc.IterationIdentifier=wt.series.IntegerSeries  
wt.vc.VersionIdentifier=wt.series.HarvardSeries  
# Series' subclassdefaults'  
wt.series.IntegerSeries.min=1  
wt.series.IntegerSeries.max=2147483647  
wt.series.IntegerSeries.delta=1  
wt.series.MulticharacterSeries.min=A  
wt.series.MulticharacterSeries.max=Z  
wt.series.MulticharacterSeries.delta=1  
wt.series.MulticharacterSeries.length=3  
wt.series.HarvardSeries.depth=16  
wt.series.HarvardSeries.level.1=wt.series.MulticharacterSeries  
wt.series.HarvardSeries.delimiter=.
```

Iteration identifiers are by default specified as being an integer series where this series contains whole numbers from a minimum to a maximum value. The default minimum is one (1) and the default maximum is the largest possible 32-bit positive integer value as defined by  $2^{*}31-1$  (2,147,483,647). There is also a default specification of the delta between adjacent numbers, which is one (1) meaning that, if given 10, the next incremented integer value would be 11 by default.

Version identifiers are by default specified as being a harvard series. A harvard series is one that is made up of subseries and is typically depicted as a stream of values delimited by a dot (for example, 1.1, 1.3.5, 1.2.1.1, and so on). Version identifiers must be specified as some sort of multilevel series where the harvard series is currently the only option. This is done to represent both in-lined versions and branched versions in one identifier that forms a version tree. In turn, the harvard series' subseries are by default specified as being multicharacter series at level one and then, because there are no other levels specified, the default from there on is what was specified before for all levels. In this case, it would be multicharacter series for all levels. If level two was specified as an integer series then, starting at the beginning, every two levels would alternate from a multicharacter to integer series up to the specified depth as seen below.

```
wt.series.HarvardSeries.depth=16  
wt.series.HarvardSeries.level.1=wt.series.MulticharacterSeries  
wt.series.HarvardSeries.level.2=wt.series.IntegerSeries
```

Multicharacter series are specified much like integer series where there is a minimum, maximum, and delta. By default, these properties specify A, Z, and 1, respectively. But there also is another property to specify the length of the multicharacter series. By default, this is three (3), which means that an identifier of this series would have a range from A to ZZZ, where incrementing from Z of a delta of one (1) would be AA, incrementing from AZ would be BA, and incrementing from ZZ would be AAA. Decrementing follows these same rules except in reverse. If a length of one (1) were specified instead of the default of three, this series would range from A to Z given the same other properties.

Any character value for the minimum and maximum can be specified for a multicharacter series. But only integers can be specified for the minimum and maximum values of an integer series.

## Customizing Indexing Behavior

You can modify the information stored in the search engine's document information table that is associated with a Windchill business object by extending the `wt.index.RwareIndexDelegate` class and providing customized behavior. The default `RwareIndexDelegate` indexes every attribute, every instance-based attribute (IBA), the identity of every object reference connected with the business object (that is, the owner), the text of content files, and the links and descriptions of attached URLs. You most likely need to modify the behavior of the `wt.index.RwareIndexDelegate` if your site has a customized object with a particular attribute that you want to appear in its own field in the search engine's index.

The following sections describe the predefined RetrievalWare fields supplied with Windchill and how to add the capability to index data into new fields.

### RetrievalWare Fields

In addition to the full text index of object metadata and content files, text can be indexed into specific RetrievalWare fields. When text is indexed into fields, it is possible to restrict, or filter, query responses by entering search terms in data fields, such as title, number, date, and so on. These queries can be used in combination with a full text query or used alone to limit the types of documents returned. Windchill provides a RetrievalWare working directory that has a file system library configured with the following fields:

Field	Description
AppOID	The object identifier of the <code>ApplicationData</code> object that this index entry represents. It is used in the construction of the URL that directly displays content files.
BusinessType	The type of the Windchill business object (document, part, change request, and so on).
Date	The date that the Windchill object was last modified.
Description	The description attribute of the Windchill object.
DocTitle	The title of the Windchill object.

<b>Field</b>	<b>Description</b>
IsAContentFile	True or false, indicating whether this index entry is a Windchill content file entry (true) or a Windchill metadata entry (false).
LifeCycleState	The life cycle state of the object.
Name	The name of the object.
Number	The number attribute of the object.
ObjectIdentifier	The object identifier of the main Windchill object that this index entry represents. It is used in the construction of a URL that displays the Windchill object's property page.
PersistInfoOID	The PersistInfo object identifier attribute of the Windchill object being indexed. It is used in the construction of the URL that directly accesses an object's primary content file.
Project	If this object's entry is associated with a project, the project name appears in this field.
StandardIcon	The path to the icon that represents the type of object that this index entry represents
SystemId	The value of the property wt.server.

If you want a library to contain additional fields, you must add those fields before data is indexed into the library. Refer to the File System Libraries section of the *RetrievalWare System Administrator's Setup Guide* for further information on setting up library fields. After the fields have been defined for the library, you can write a custom `RwareIndexDelegate` to populate the new fields with the appropriate data.

## Setting the Length of Path Entries

By default, path entries for objects entered in the database are set to 40 characters. To use entries longer than 40 characters, increase the `SIZE` variable of the `SystemId` field in the file `<RWare>\windchill_indexes\config\rware.cfg`.

The initial setting of `SystemId` in `rware.cfg` is shown in the following code segment (long lines wrap around for presentation in the manual):

```
FIELD "SystemId"{  
INFO "Windchill system id that requested this object to be indexed.  
    Used to construct a URL that will go to the appropriate Windchill  
    system to retrieve the Windchill object.";  
SIZE 40;  
TYPE STRING;  
FLAGS STORED INDEXED VISIBLE TOKENS_ONLY USER_SEARCH WITH_BODY;
```

Edit the `rware.cfg` file to set the `SIZE` variable to an integer value that will accommodate the length of the desired paths.

Anytime you change this setting, you must use the RetrievalWare System Utilities to delete and create a new index for the libraries, and repeat the `BulkIndex` operation on them. Failure to delete and create the index, and then re-index (using `BulkIndex`) will result in an error in the RWare Executive server and it will not start.

To perform the required actions after changing the `SystemId` property, use the following procedure:

1. Start the RetrievalWare System Utilities Menu.
2. If the Administration Servers or Search and Indexing Servers are already running, shut them down as follows:
  - a. Select Administration Servers (option 3).
  - a. Select Shutdown (option 5).
  - a. Select [ Return to previous menu ] (option 1).
  - a. Select Search and Indexing Servers (option 4).
  - a. Select Shutdown (option 5).
  - a. Select [ Return to previous menu ] (option 1).
3. Select Indexing and index utilities (option 6).
4. Select Delete and create empty indexes for a library (option 10).
5. Enter the applicable library's name.
6. Respond Yes or No when the following prompt appears:

Are you sure?

7. When the script is finished, the following prompt appears:  

```
"Press any key to continue . . ."
```
8. Repeat steps 4 through 7 for all new libraries you want to index.
9. Select [ Return to previous menu ] (option 1).
10. Select [ Return to previous menu ] (option 1).
11. Restart the Administration Servers and Search and Indexing Servers.
  - a. Select Administration Servers (option 3).
  - b. Select Start servers in background (option 2).
  - c. Select [ Return to previous menu ] (option 1).
  - d. Select Search and Indexing Servers (option 4).
  - e. Select Start servers in background (option 2).
  - f. Select [ Return to previous menu ] (option 1).
12. Instantiate `java wt.index.BulkIndexTool` and re-index the affected libraries.

## Creating a Custom `RwareIndexDelegate`

The default `RwareIndexDelegate` implementation indexes all object metadata and any instance based attributes (IBAs) associated to the current object. Creating a custom `RwareIndexDelegate` object is appropriate if you know which fields you want indexed for an object and you only want those fields indexed. This is the primary reason for creating a custom `RwareIndexDelegate`. A custom `RwareIndexDelegate` could theoretically improve indexing performance because fewer attributes would need to be processed and written to `RetrievalWare` but this should be a secondary consideration when determining if creating a custom `RwareIndexDelegate` is appropriate.

To create a custom `RwareIndexDelegate`, use Rational Rose to model your new class, making it extend the `RwareIndexDelegate` class. Then generate the Java code and provide implementations for one or more of the following method signatures:

```
public void custom_getMetaData()  
  
    throws WTEException, IndexingException {
```

Uses introspection to get all attributes and their values defined for the current `Indexable` object's type and gets all IBAs and their values for the current object and indexes them to `Rware`. This method must be overwritten to do nothing or return a specific set of fields if custom behavior is desired.

```
public void custom_getAdditionalMetaData()  
  
    throws WTEException, IndexingException {
```

This customization point should not be used. All metadata customizations can be handled in `custom_getMetaData()`.

```
public void custom_getFieldData()  
    throws WTEException, IndexingException {
```

Gets a default list of attributes like name, lifecycle state, modifyTimestamp, etc. and their values and indexes them to Rware. If the default implementation for `custom_getMetaData()` is being used then this method is not necessary because any processing it does will be duplicate to what `custom_getMetaData()` does. The only customizations that are valid for this method if `custom_getMetaData()` has been customized are for attributes that are not detectable through introspection or as IBAs.

```
public void custom_getAdditionalFieldData()  
    throws WTEException, IndexingException {
```

Customization point that allows for specific extra attribute and their values to be defined and indexed to Rware. This customization point should not typically need to be implemented.

In previous releases there were restrictions on removing the AppOID, ObjectIdentifier, PersistInfoOID, and SystemId attributes. There are no such restrictions in 7.0, customizers do not need to index these attributes. In 7.0, PersistInfoOID is the only required attribute and it is handled outside of the `custom*` methods so customizers do not need to be concerned with it.

To put your custom `RwareIndexDelegate` into action, you must update the `service.properties` file to reflect this change. Following is the default `RwareIndexDelegate` entry:

```
# #####  
# The wt.index.IndexDelegate service.  
# #####  
# note that you can change the "null" here for selector to a  
# collection name to map a particular collection  
# to a particular IndexDelegate  
wt.services/svc/default/wt.index.RwareIndexDelegate/null/  
wt.index.Indexable/0=wt.index.RwareIndexDelegate/singleton
```

For this example, assume that you have a custom class, `mycust.MyChangeRequest`, and have created a custom `RwareIndexDelegate` named `mycust.ChangeRequestIndexDelegate` that you want to use to index objects of this type. If you want to override the default `RwareIndexDelegate` for just the class `mycust.MyChangeRequest` and its subclasses with the `mycust.ChangeRequestIndexDelegate`, you must add the following line to `service.properties`:

```
wt.services/svc/default/wt.index.RwareIndexDelegate/Windchill_
  Business_Collection/mycust.MyChangeRequest/0=
  mycust.ChangeRequestIndexDelegate/singleton
```

(Note that you should actually enter only one line; the indentation shown here indicates a continuation of the preceding line, necessary for presentation of lines within the book.)

Alternatively, you may want all objects to go to a particular collection to be indexed using a customized `RwareIndexDelegate`. In this case, you need to make an entry similar to the following:

```
wt.services/svc/default/wt.index.RwareIndexDelegate/
  A_Collection_Name/wt.index.Indexable/
  0=mycust.ACustomIndexDelegate/singleton
```

In this example, `A_Collection_Name` is a collection from the comma-separated list of collections assigned to the property `wt.index.collections`, and `mycust.ACustomIndexDelegate` is the class that should be used when indexing an object to `A_Collection_Name`. A different subclass of `RwareIndexDelegate` can be used for different collections by putting a separate entry in `service.properties` for each collection.

## Overriding the `custom_getAdditionalFieldData()` Method

The `custom_getAdditionalFieldData()` method can be overridden similar to the following to populate additional fields of a `RetrievalWare` library:

```
public void custom_getAdditionalFieldData()
    throws WTEException, IndexingException {
    // Use the getIndexable() method to get the object that is
    // being indexed.
    Object customObject = getIndexable();

    // Then use whatever API calls are needed to get the data
    // that is to populate the fields.
    String fieldValue1 = customObject.getFieldValue1();
    String fieldValue2 = customObject.getFieldValue2();

    // Finally, send the field name and value to the indexloader
    // using the IndexAccessor writeField method.
    getIndexAccessor().writeField("NewField1", "fieldValue1");
    getIndexAccessor().writeField("NewField2", "fieldValue2");
}
```

## Overriding the `custom_getAdditionalMetaData()` Method

The `custom_getAdditionalMetaData()` method can be overridden similar to the following to provide additional metadata or text to be indexed:

```
public void custom_getAdditionalMetaData()
    throws WTEException, IndexingException {
    // Use the getIndexable() method to get the object that is
    // being indexed.
    Object customObject = getIndexable();
```

```
// Use the getIndexAccessor() method to get the indexAccessor
// and then use the writeMetaData() method to write custom
// metadata information.
getIndexAccessor().writeMetaData(
    "a string of metadata - obtained from the object");
}
```

## RwareIndexDelegate Method Reference

Following are RwareIndexDelegate methods:

### **public Indexable getIndexable( )**

This method returns the object that is the target of the event that triggered the indexing service to begin indexing the object.

### **public IndexAccessor getIndexAccessor( )**

This method returns an initialized IndexAccessor object that can be used to write fields and metadata to the indexloader. See the IndexAccessor Method Reference section that follows for further information on the writeField() and writeMetaData methods of this class.

## IndexAccessor Method Reference

Following are IndexAccessor methods:

### **public boolean writeField (String field\_name, String value)**

This method sends the String "value" to the indexloader to be indexed into the field identified by the String "field\_name".

### **public boolean writeMetaData (String data)**

This method sends the String "data" to the indexloader to be indexed as metadata.

## Information Transfer When Indexing a Windchill Business Object

When the IndexPolicyManager determines that an object must be indexed, it requests an RwareIndexDelegate from the IndexDelegateFactory to get the appropriate RwareIndexDelegate for the object class and collection. It then invokes the RwareIndexDelegate's index() method. In previous releases, the indexed information was written to a temporary .wsf file and then to RetrievalWare but in 7.0 the indexed information is written directly to RetrievalWare. You can see what information is being indexed by setting wt.index.verboseExecution=true in your wt.properties file. The debug output from IndexEntryToRware will be similar to the following::

```
<000121Windchill>
<fields>
field PersistInfoOID wt.doc.WTDocument:33843
field AppOID wt.content.ApplicationData:33851
field BusinessType Document
field Date 4/4/00 4:08 PM
field Identity 22 - GadQRef, A
field Number 22
field ObjectIdentifier VR:wt.doc.WTDocument:33840
field NewField2 fieldValue2
field NewField1 fieldValue1
field docTitle Gadget Quick Reference Guide
field Name GadQRef
field StandardIcon wt/clients/images/msword.gif
field LifeCycleState Released
field SystemId windchill
```

```

</fields>
Modifier Name hej Creator Full Name Helen Ellen Jerimico Modifier EMail
demo_user Branch Identifier 33840 Format Name Microsoft Word Type
Document Department Documentation Last Updated 4/4/00 4:08 PM Cabinet
Publications Identity 22 - GadQRef, A Number 22 Folder Path
/Publications/GadQRef Creator Name hej Modifier Full Name Helen Ellen
Jerimico Created 4/4/00 4:08 PM State Released Business Type WTDocument
Title Gadget Quick Reference Guide Location /Publications Conceptual
Classname wt.doc.WTDocument Type Document Type Document Life Cycle Released
Data Name GadQRef Creator EMail demo_user a string of metadata - obtained
from the object Primary Microsoft Word GadQRef.doc
<file><title>GadQRef.doc</title><appOID>wt.content.ApplicationData:33851
</appOID>c:empcontentFile2.bin</file>

```

This example actually represents what the debug output would look like if the `RwareIndexDelegate` was extended and the `custom_getAdditionalFieldData()` method was overridden as in the preceding example (note the two new fields in bold). Assuming that the library definition defined `NewField1` and `NewField2`, it would be possible to search over these new fields.

After the `IndexEntryToRware` class generates this debug output, it makes an RPC call to the `RetrievalWare` server telling the server to index it. The `RetrievalWare` server then uses the document parser command file associated with the library that the document is to be indexed into to parse the file.

Windchill supplies a document parser command file named `wb_lib.dp`, which parses files of the preceding format. This file is located in the `windchill_indexes` working directory (`./dp/wb_lib.dp`). The parser loops through the fields and takes the first word after "field" as the field name and uses the rest of the line as the field value.

Everything after the `</fields>` tag and before the `<file>` tag is indexed as the document body. Overriding the `custom_getAdditionalMetaData()` method adds to this text while overriding the `custom_getMetaData()` method. The `<file>` and `</file>` tags are used to describe one or more content files. The parser sees these and tells `RetrievalWare` to process them as children of the document that is currently being indexed.

## Customizing the Enterprise Search Interface

The Enterprise Search interface was developed using the RetrievalWare Web Tool Kit. You can customize it or develop a new interface using this toolkit. Refer to the *RetrievalWare Web Toolkit Guide* for further information on developing user interfaces.

## Customizing a Bill of Materials

### Overview

A Bill of Materials (BOM) is a product structure report that identifies the items required to assemble a product. Currently, the BOM report includes only items from a part structure. Two types of BOMs are supported.

- A BOM Hierarchy report shows the quantity and unit attributes of the "usage" associations and visually indicates the hierarchical levels.
- A Parts List report is a list of all of the components (that is, parts that do not use other parts) in the part structure hierarchy. It uses information in the "usage" associations to determine the quantity of each component. Basically, it answers the question "When a part assembly is completely disassembled, what parts and how many of each do I have?".

Both of these reports are defined to be generated as an HTML page optimized for printing from a Web browser. Both the BOM generation and formatting can be customized as needed.

A BOM report is generated using a template processor. The actual BOM information is generated entirely by Visitor objects. These objects are a means of specifying a "callback" operation for each node in a "tree" data structure. For example, a part structure contains Part objects linked together through the "uses" relationship. Starting from a top-level part, the entire part structure is traversed and a "visit" method is called on the Visitor object passing the current part object and the related usage information. This method can be overridden to provide customized BOM computation and formatting.

### Customization

The remainder of this section describes an example of a BOM that is provided out-of-the-box with Windchill. Understanding this example provides a basis for implementing other BOM customizations. There are two types of customization for a BOM. BOM computation refers to the work that is done on each item as the structure is traversed. Windchill provides two types of BOM computation: hierarchy and parts list. The second customization is for formatting the output that is displayed to the user. This formatting is dependent on the type of information displayed.

The Hierarchy Visitor is an implementation that shows the quantity and unit attributes of the "usage" associations and visually indicates the hierarchical levels. The only computation that is required is to output this information. This class extends the TextOutputVisitor, which provides standard helper methods for outputting localizable messages to an output stream. In addition to providing the implementation for the "visit" method, this class defines the customization points for altering the format of the output. These methods are designed to be called at various points as the output is being generated.

```
public abstract class BOMHierarchyVisitor extends TextOutputVisitor {

    // Define methods that should be overridden to allow for
    // formatting customization

    protected abstract void preVisitWrite( int a_level );

        protected abstract void postVisitWrite( int a_level );

    public void preNavigateWrite() {
    }

    public void postNavigateWrite() {
    }
}
```

The visit method performs a basic output of the part identity and usage information. The preVisitWrite() and postVisitWrite() calls are significant because they allow for formatting customization.

```
public boolean visit( Persistable fromNode, Link link,
    Persistable toNode, int level,
    boolean previouslyVisited ) throws WTEException {

    PrintWriter out = getPrintWriter();

    preVisitWrite(level);

    String partIdentity = ((WTPart) toNode).getDisplayIdentity();
    if(link == null) {
        out.print(partIdentity);
    } else {
        // Display part and usage information as a
        // localized message
        Quantity quantity = ((WTPartUsageLink) link).getQuantity();
        Object[] params = { new Double(quantity.getAmount()),
            quantity.getUnit().getDisplay(), partIdentity };
        printLocalizedMessage(partResource.PART_USAGE, params);
    }

    postVisitWrite(level);

    return true;
}
```

The BOMHierarchyVisitor class is abstract and therefore cannot be used directly. The formatting must be customized by implementing the preVisitWrite() and postVisitWrite() methods in a descendant class. The following class is a customization example for use when the BOM is displayed in an HTML page. It displays the hierarchical level through HTML heading tags. The methods preVisitWrite() and postVisitWrite() wrap the part information in and tags where n is the level number.

```
public class HtmlHeadingHierarchyVisitor extends BOMHierarchyVisitor {

    protected void preVisitWrite( int a_level ) {

        PrintWriter out = getPrintWriter();

        out.print("&lt;H");
        out.print(a_level + 1);
        out.print('>');

    protected void postVisitWrite( int a_level ) {

        PrintWriter out = getPrintWriter();

        out.print("&lt;/H");
        out.print(a_level + 1);
        out.print('>');

    out.flush();
    }
}
```

After the customization is complete, you need only specify that the new customized class should be used when the BOM is generated. By default, the BOM template processors use the application services mechanism to instantiate a Visitor object that is used to generate the BOM. The service can be replaced by changing the appropriate entry in the services.properties file. Change the following entry in the file:

```
wt.services/svc/default/wt.part.BOMHierarchyVisitor/print/  
wt.fc.WTObject/0=wt.part.HtmlNumberedHierarchyVisitor/duplicate
```

to:

```
wt.services/svc/default/wt.part.BOMHierarchyVisitor/print/  
wt.fc.WTObject/0=wt.part.HtmlHeadingHierarchyVisitor/duplicate
```

Note the use of the "print" value in the "visitor" key in the service entry. Currently, the Windchill client code displays a BOM designed for printing within an HTML page and therefore uses the value "print". This key value is obtained from the URL parameters by the BOM template processor. If the client were customized to provide other types of BOM formats (by specifying different values in the URL), this key could be used for generating different formats. For example, an additional client function could be added for viewing a BOM with embedded graphics. A new value "graphic" could be used that is associated with a customized Visitor that will generate appropriate graphic output. Both types of formatting would then be available to the user.

## Current Implementation of Visitors

The information in this section is useful for understanding the implementation of the current out-of-the-box Visitors.

The computation of the Parts List BOM (in `BOMPartsListVisitor`) uses two separate internal data structures to store information about the part structure as it is traversed. An assembly table keeps track of the current quantity and unit for each subassembly part as successive levels are encountered. A component table keeps track of the nonseparable parts (leaf nodes in the structure) that become the parts list. The assembly table information is maintained by multiplying the parent part's quantity and the current part's quantity. For example, if part A uses 3 of part B, which uses 5 of part C, then 15 of part C are required. The component table stores information for the actual parts that make up an assembly. This is used when generating the final report on the parts list. It is possible that a part can be used in separate branches in the part structure. In these cases, the quantity in each branch should be added. To continue the example, if part A uses 1 of part D, which uses 6 of part C, then the total number of part C for the entire part structure is 21  $((3 * 5) + (1 * 6))$ .

A related issue is how to handle processing the quantities with respect to the quantity units. The following rules are applied:

- If the units are identical, the new unit is the current unit.
- If either unit is `AS_NEEDED`, the new unit is `AS_NEEDED`.
- If either unit is `EACH`, use the other unit.

Both the Hierarchy and Parts List Visitor objects are dependent on a depth first traversal of the part structure. The Hierarchy Visitor outputs the structure as it is traversed so the order in which part nodes are visited is significant. The Parts List computation relies on multiplying quantities between levels and storing the result internally. If a part were to appear in separate branches of a structure and depth first traversal were not used, then the computed quantity for a subassembly can be overwritten before its component parts use that information.

# Customizing User Preferences

Detailed reference information on user preferences can be found in the *Windchill Application Developer's Guide*. The section discusses and gives examples of how you customize the out-of-the-box user preference functionality.

## The Preference Hierarchy and Delegates

### Preference Key Uniqueness

The preferences framework is based on the principal that a unique preference consists of the following attributes:

- Parent node (or root node, if at the top of the hierarchy)
- Preference node (usually associated as a group of similar preferences)
- Preference key

Together these attributes form a unique key structure of parent/node/key. This unique key structure is referred to as the *fully qualified preference key*. To separate individual user and group preferences for the same fully qualified preference key, a context is applied to the preference.

The context consists of the following elements:

#### **Macro**

A constant defining the type of context (see the section on preference macros that follows).

#### **Descriptor**

Optional text defining the name of the context.

These elements are connected by a colon (':') to form the preference context.

When combined with a context, the fully qualified preference key forms a unique row in the database table, thus allowing users and other preference divisions to have individual preferences.

### Preference Macros

The `wt.prefs.WTPreferences` class defines the following types of preference context macros:

**USER\_CONTEXT** - the context for individual users.

**DEFAULT\_CONTEXT** - the context for the system default (shipping) values.

**CONTAINER\_CONTEXT** - a context used in the container hierarchy.

**CONTAINER\_POLICY\_CONTEXT** - a container context that is enforced as a policy.

**DIVISION\_CONTEXT** - the context used for any scopes defined in addition to the default, container, and user scopes.

**DIVISION\_POLICY\_CONTEXT** - a division context that is enforced as a policy

## Setting the Hierarchy

The `wt.prefs.delegates.DelegateOrder` controls the hierarchy in which delegates are called. For each level in the hierarchy there should be an entry in this property. The customized entries should appear as **DIVISION\_CONTEXT**. For example, in the out-of-the-box hierarchy, there is a division scope called Windchill Enterprise, and the out-of-the-box `wt.prefs.delegates.DelegateOrder` property value is:

```
$DEFAULT,$CONTAINER,$DIVISION:WindchillEnterprise,$USER
```

In this value, there is no **DIVISION\_POLICY\_CONTEXT** defined since **DIVISION\_POLICY\_CONTEXT** and **DIVISION\_CONTEXT** are related and are at the same level in the preference hierarchy. Similarly, the **CONTAINER\_POLICY\_CONTEXT** need not be included. Entries are designated differently only when storing and retrieving preferences internally. For more details on correctly naming delegates, see the `wt.prefs.delegates` file.

If `wt.prefs.delegates.DelegateOrder` has been removed from the `wt.prefs.delegates` file, Windchill uses the following:

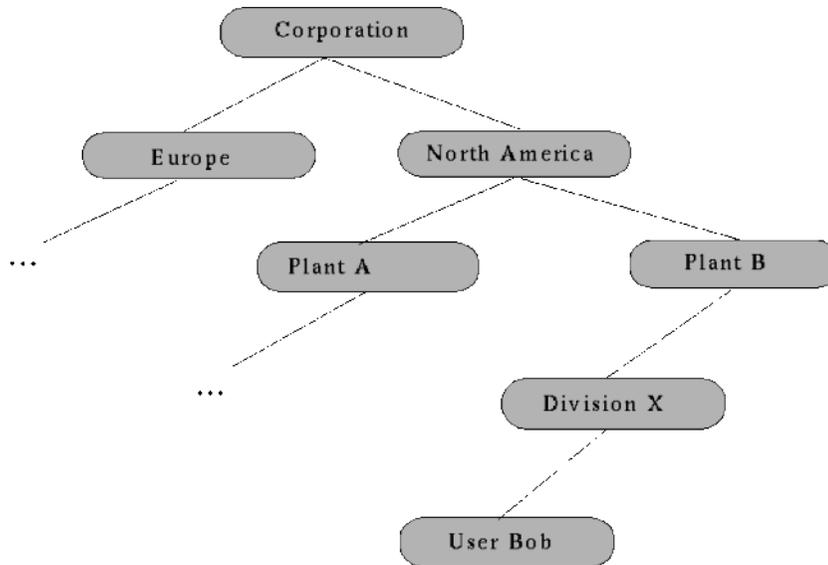
```
$DEFAULT,$CONTAINER,$USER
```

The preference framework was designed so the **USER** and **DEFAULT** contexts would be the top and bottom level of the preference hierarchy. In this case, the `wt.prefs.delegates.UserDelegate` and `wt.prefs.delegates.SystemDefaultDelegate` can be used out of the box without any modification. The rest of the hierarchy should be handled by a customized preference delegate as described in the remainder of this section.

## Preference Hierarchy

An important distinction to make is the distinction between the preference hierarchy and the preference structure. The preference structure is the data structure used to store preferences and it is implemented by the preferences framework. The preference hierarchy is the hierarchy used to describe which preferences take precedence over other preferences. For example, assume the following global corporate hierarchy for a company named The Acme Corporation.

The Acme Corporation consists of two regional divisions: Europe and North America. Each region can have a variable number of plants. Each plant can have a variable number of divisions. Finally, each division consists of various users, all as illustrated in the following figure.



This example is a simplified corporate structure, but it illustrates how the preference hierarchy behaves. Also, in this example, the structure is represented as a tree when, in reality, it may take on many different structures. The hierarchical levels in the structure are managed by *preference delegates*. In this example, the bottom level in the hierarchy (known as leaf nodes) is managed by the `wt.prefs.delegates.UserDelegate`, which implements the macro `USER_CONTEXT`. The top level (above Corporation) is the `DEFAULT_CONTEXT` level, which is managed by the `wt.prefs.delegates.DefaultSystemDelegate`. Both of these delegates can be exchanged for a customized delegate; however, the customized delegate must implement `DEFAULT_CONTEXT` and `USER_CONTEXT` in order for the preferences framework to operate properly. (Details on the implementation of these delegates can be obtained from their javadoc.)

To handle the rest of the hierarchy, you must write a delegate that extends `PreferenceDelegate` and implements its abstract methods. The number of delegates that perform this task can vary (for example, each level may have its own delegate) or a single delegate can be used to handle several levels of the tree.

Adding a delegate to the preference hierarchy requires the following two steps:

1. Add the delegate in the correct order in the `wt/prefs/delegates/delegates.properties` file.
2. Add the delegate class that represents the hierarchical level in the `delegates.properties` file.

### Properties Value

The `wt/prefs/delegates/delegates.properties` value **`wt.prefs.delegates.DelegateOrder`** is the property that controls the hierarchy in which delegates are called. For each level in the hierarchy, there should be an entry in this property. Customized entries should appear as

DIVISION\_CONTEXTs. For example, for a Division named "WindchillEnterprise", the property value should be as follows:

```
$DEFAULT,$DIVISION:WindchillEnterprise,$USER
```

Note that no DIVISION\_POLICY\_CONTEXT is defined because, as far as delegates are concerned, a Division preference and a Division policy preference are the same thing. They are designated differently only when storing and retrieving preferences.

A property that corresponds to the delegate must also be added. The property name should be as follows:

```
wt.prefs.delegates.class.$MACRO
```

where \$MACRO is the name of the preference context, for example, \$USER\_CONTEXT. The value should be defined as the name of the preference delegate to invoke to manage this context. For implementation details, see the delegates.properties file.

## Delegate Roles

All preference delegates should be extended from the abstract class `wt.prefs.PreferenceDelegate`. This class defines the methods that each delegate must implement within the preferences framework. These methods are as follows:

- `public String getLocalizedString(String division, Locale aLocale)`
- `public boolean isAdministrator(String division, WTUser user)`
- `public ArrayList getDivisionsAsAdministrator(WTUser user)`
- `public ArrayList getDivisions(WTUser user)`

The `getDivisions` methods differ in that the `getDivisionsAsAdministrator(user)` method should return a list of all the levels in this delegate for which the given user is responsible or able to administrate. The `getDivisions(user)` method should return all the divisions of which the given user is a member.

Using the figure of the corporation shown earlier, if this was a delegate that managed the tree from Corporation to divisions, a user named Bob is a member of Division X, Plant B, North American and Corporation. Internally, how the delegate decides which users are members or administrators of divisions is up to the developer. The preferences framework is strict in that it expects a well-formed answer to the preceding methods. If a user is not a member of any division managed by the delegate, then the `ArrayList` should be empty, not null. For details on the implementation of the `wt.prefs.delegates.*` classes, see their javadoc.

## Writing a Servlet Helper to Use URLFactory Functionality

Servlets within Windchill are accessed by requesting a Web resource that exists under a path with the following format:

*<Web application>/servlet/<your servlet name>*

The role and arguments of servlets can vary greatly depending on their application. However, if HREFs and URLs are being generated to the servlet, the URLFactory should be used in conjunction with a servlet helper to add the functionality provided by the URLFactory.

The GatewayServletHelper class located in wt.httpgw is a good example of how to create HREFs and URLs to the anonymous and authenticated gateway servlets. A preconfigured URLFactory is taken as the first argument of each method, and this URLFactory is used to create the final HREF or URL accordingly within the method.

When writing servlet helpers, keep the following points in mind:

- All methods should have a URLFactory as their first argument. (See the example that follows.)
- URLFactory may be null or uninitialized. In this case, the method body should instantiate a new URLFactory for the HREF and URL generation.
- The method name should follow either of the following helper conventions, where *<servlet>* is a name representing the servlet:
  - **build<servlet>HREF()**
  - **build<servlet>URL()**
- The method bodies should take arguments and generate a proper resource path (such as, *servlet/WindchillAuthGW*), then pass this resource path to the URLFactory *getHREF* and *getURL* methods for generating the servlet HREF and URL.

The *Windchill Application Developer's Guide* gives more detailed reference information on URLFactory usage and encoding/decoding techniques to use in JSP development.

## Example URLFactory Implementation

Following is a sample file setting up a default URLFactory implementation in an internationalization environment.

```
<%
// bcwti
// Copyright (c) 2001 Parametric Technology Corporation, all rights reserved.
// ecwti

////////////////////////////////////
// This file is intended to give an example of how the URLFactory
// and GatewayServletHelper may be used within
// the JSP programming environment. To run this file, simply
// place the file in the /Windchill/codebase/wtcore/jsp directory.
// Extra whitespace as been left in for clarity and ease of reading.
////////////////////////////////////
%>

<% /** The WtContextBean is a JavaBean for use in Java Server Pages or
    Servlets that wish to use Windchill Java client or server APIs */ %>

<jsp:useBean id="wtcontext" class="wt.httpgw.WtContextBean" scope="request">
  <jsp:setProperty name="wtcontext" property="request" value="<%=request%>" />
</jsp:useBean>

<% /** The URLFactory is a JavaBean for use in the generation of HREFs used in
    Windchill JSP clients */ %>
<jsp:useBean id="url_factory" class="wt.httpgw.URLFactory" scope="request" >
<%

url_factory.setRequestURL(request.getScheme(),request.getHeader("HOST"),
  request.getRequestURI());
%></jsp:useBean>

<%
// The response header for the output stream should be set to UTF-8
response.setContentType("text/html; charset=UTF-8");
// Below we also set the content type in the @page tag, since some
// servlet engines
%><@ page import="wt.httpgw.*,java.util.HashMap" contentType="text/html;
charset=UTF-8" %>

<%
// Here we want to generate a link to the Windchill home page through
// the default authenticated gateway. For this we will utilize the
// wt.httpgw.GatewayServletHelper class to generate a HREF which
// from the current URLFactory will generate a link to this site.
// %>
<BR>
<A HREF="<%=
wt.httpgw.GatewayServletHelper.buildAuthenticatedHREF( url_factory )
%>">Windchill Home</A>
<BR>
<% // Perhaps you would like a link to your personal cabinet. In order to
// do this, you must generate a link through the authenticated gateway
// with a class/method and arguments to invoke like below
```

```

//
String URL_PROCESSOR = "wt.enterprise.URLProcessor";
String TEMPLATE_ACTION = "URLTemplateAction";
// URLFactory and GatewayServletHelper uses HashMap for arguments
HashMap map = new HashMap( );
map.put( "Action", "Create" ); // adds the action 'create' to the
hashmap
map.put( "Class", "wt.doc.WTDocument" ); // adds the class to apply the action
// to.

%>
<A HREF="<%=
wt.httpgw.GatewayServletHelper.buildAuthenticatedHREF(url_factory,
URL_PROCESSOR, TEMPLATE_ACTION, map )
%>">Create a Document</A>

<%
// For optimization, any links generated in the fashion described above could be
// generated reusing the HashMap and Strings already created. The Setup.jsp file
// located in codebase/wtcore/jsp/wt/portal/Setup.jsp does just this in the
// generation of the links thus reducing the overhead in String/HashMap object
// creation.
//
// In the FORM below we use the URLFactory to generate a link to the resource
// wtcore/jsp/sample.jsp. See the javadoc for method prototypes. The URLFactory
// is smart enough to see how it is currently configured and return a String link
// which should work within the current environment. For the example below, since
// we are running the file from /WEB-APP/wtcore/jsp the link will generate simply
// "sample.jsp". Optionally we could have called
// url_factory.getHREF("wtcore/jsp/sample.jsp",request.getQueryString() ) if we
// wished to retain the current Query string.
%>
<FORM ACTION="<%= url_factory.getHREF("wtcore/jsp/sample.jsp") %>" METHOD="POST">
<INPUT TYPE="text" NAME="Sample_Text" VALUE="<%= prev_value %>">
<INPUT TYPE="submit">
</FORM>

```

# 4

## Customizing GUIs

<b>Topic</b>	<b>Page</b>
Customizing Windchill Portal Pages.....	4-2
Customizing Colors in HTML and JSP Clients .....	4-5
Customizing Localizable Display Names .....	4-10
Customizing Local Searches .....	4-12
Customizing the HTML Search .....	4-17
Customizing the Windchill Explorer.....	4-23
Customizing Objects Created from the Windchill Explorer .....	4-31
Making Parts Content Holders .....	4-32
Adding URLs to the Method Server.....	4-32

# Customizing Windchill Portal Pages

The out-of-the-box Windchill home pages, also referred to as *portal pages*, are deployed using JavaServer Page (JSP) technology. Because each portal page is a separate JSP page, they are relatively easy to customize. The following files work together to produce these pages. Edit these files to change the appearance and functionality of your site's portal pages.

## **index.jsp**

This is the beginning of the portal pages. It contains references to the `header.jsp` and `footer.jsp` files, which control the appearance of the header and footer, respectively, on all portal pages.

The `header.jsp` file contains the PTC logo in the image file `logo.gif`; you will probably want to change this at your installation. You can either replace this file in the `portalimages` folder or edit the `header.jsp` file to point to a different logo image.

## **WTDefault.css**

This file also controls the appearance of portal pages. The definitions in this file control the font, and the font's colors and sizes. It is designed so that any change to this file is propagated to all the portal pages.

This file is also used in some Windchill Foundation HTML clients. For further information, see the section on Customizing Colors in HTML and JSP Clients later in this chapter.

## **setup.jsp**

This file contains most of the Java code that provides the functionality for portal pages. It primarily creates strings for URLs so that the portal pages can link to other parts of the Windchill system. It also handles the processing of locales so that most text is returned in the correct language.

All `.jsp` pages include `setup.jsp` at the top of the file. This content gives the `.jsp` pages access to the variables defined in this file (mostly strings). Once these variables are included, the other pages can make use of them, such as for printing out a link to the worklist, as shown in the following example:

```
<%= worklistlink %>
```

## **sethome.jsp**

This file is called when users click on a link to go to the portal page they want set as their personal home page (that is, the page they return to when they click **Home**). It takes the **inc** and **body** arguments passed to it and sets these values in the user's preferences.

The **inc** argument determines the links on the left of the user's home page and the **body** argument identifies the page that makes up the body of the home page. For most pages, the body is set to the `search.jsp` page. To set your home page to have a different body, create a link to `sethome.jsp` with `body=newbody.jsp`, where `newbody.jsp` is the file you want included as the body of your home page. The same process applies to specifying links using the **inc** argument.

### **extras.jsp and clearhome.jsp**

These files are not linked to from the portal pages but are used mainly for debugging purposes. The most useful feature of `extras.jsp` is that it has a link to the pre-Release 5 Windchill home page (the page displaying icons rather than links). It also displays how Windchill preferences are set and gives you a link to `clearhome.jsp` for clearing your preference settings.

### **search.jsp**

This page is included in every subportal or home page. It gives all portals quick access to local search.

### **common.jsp**

This file contains the links to the subportal pages. To create a new subportal page, add the link to the new page to this file. This file is included in the left-hand navigation (purple) bar of the common home page and the site map home page.

### **commonbody.jsp**

This file contains the body of the first page all Windchill users see. This is the best place to put personalized instructions on how you have set up portal pages at your site and information to help your users pick a portal.

The remainder of the `.jsp` files are sets of links that are available for their particular subportal home page.

When users first enter Windchill, they see a page referred to as the *common home* or *common portal*. From the common home, users should pick from the available portal pages and set one of them as their new home page.

Users see the common home page first because they do not yet have any preferences set and, therefore, `index.jsp` uses its built-in defaults for the arguments **inc** and **body**. The default for **inc** is `common.jsp` and the default for **body** is `commonbody.jsp`. When users click on one of the available home links to set their default home page, information is passed in the URL to the `sethome.jsp` file about the **inc** and **body** that they want incorporated into their home page. You can change these links or add to them to allow more choices for home pages.

Once a user sets preferences by following a link to `sethome.jsp`, the `index.jsp` page can look up the user's **inc** and **body** values each subsequent time the user enters Windchill and create the page set as the home page. This can be overridden by calling `index.jsp` with the arguments **inc** and **body** defined differently. For example, suppose in the location area of your Web browser, you entered the following:

```
http://myserver/index.jsp?inc=common.jsp&body=commonbody.jsp
```

Following such a link would cause the `index.jsp` page to create the common home page despite the fact that you may have **inc** and **body** defined differently in your preferences.

If you called `index.jsp` with only one of the arguments specified, the value currently set for the omitted argument would be used as the default. For example, suppose you entered the following:

```
http://myserver/index.jsp?body=extras.jsp
```

In this case, the body of the page created would be the `extras.jsp` page, but the links on the left-hand side would be whatever is set in your **inc** preferences.

## Customizing Colors in HTML and JSP Clients

The colors of ProjectLink and PDMLink JSP clients are derived primarily from the stylesheet `<WT_HOME>/codebase/netmarkets/css/nmstyles.css`. See the style sheet for more information.

The colors of PDMLink template processing pages are derived from the following stylesheets:

- `<WT_HOME>/codebase/netmarkets/css/nmstyles.css` -for page headers, tabs and footers.
- `<WT_HOME>/com/ptc/core/ui/solutions.css` - for page body

The colors of most Windchill PDM HTML and JSP clients are set using either of the following:

- The default Windchill stylesheet named `WTDefault.css`, which is located in `Windchill/codebase`.
- Color properties in `Windchill/codebase/wt.properties`.

The color properties in `wt.properties` are most commonly used.

Each color property has the following format:

**`wt.html.color.<style class>`**

For example:

```
wt.html.color.bg-body=#FFFFFF
wt.html.color.bg-msg=#DFDFDF
wt.html.color.bg-navbar=#DFDFDF
wt.html.color.bg-pagetitle=#066699
```

The style class referred to is one in the `WTDefault.css` stylesheet, and the color values are identical to those in the stylesheet for each class.

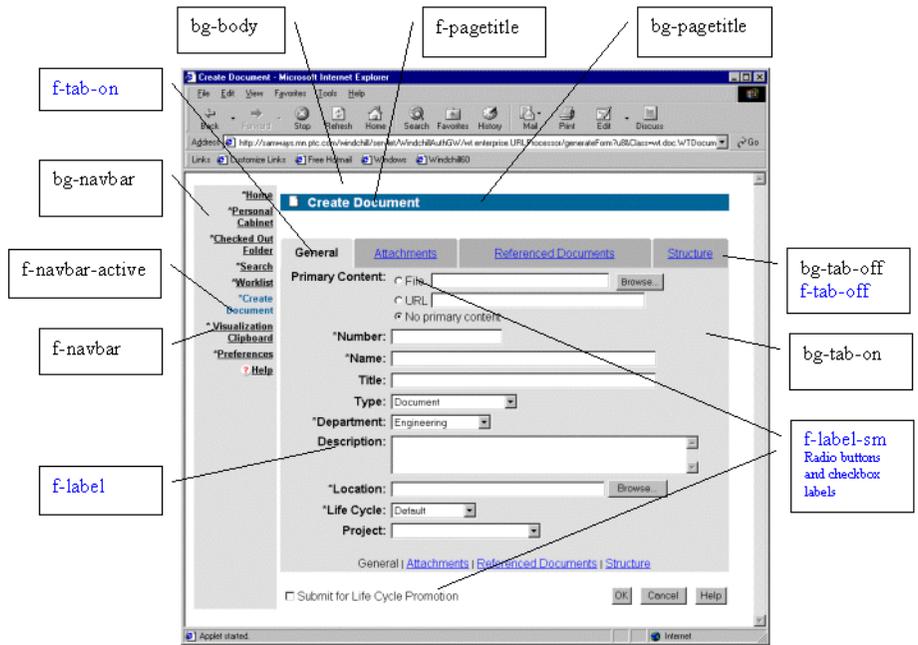
You can change the colors in most HTML and JSP Windchill PDM clients by modifying the color values in `Windchill/codebase/wt.properties` and `codebase/WTDefault.css`. If you are using the Windchill Visualization features, you must also modify the file `Windchill/codebase/wtcore/jsp/wvs/style.jsp`.

Typically, non-Foundation clients still use hardcoded color values in their HTML templates. Modifying the colors of such clients requires modification of individual templates. (For further information, see chapter 7, Customizing the HTML Client.)

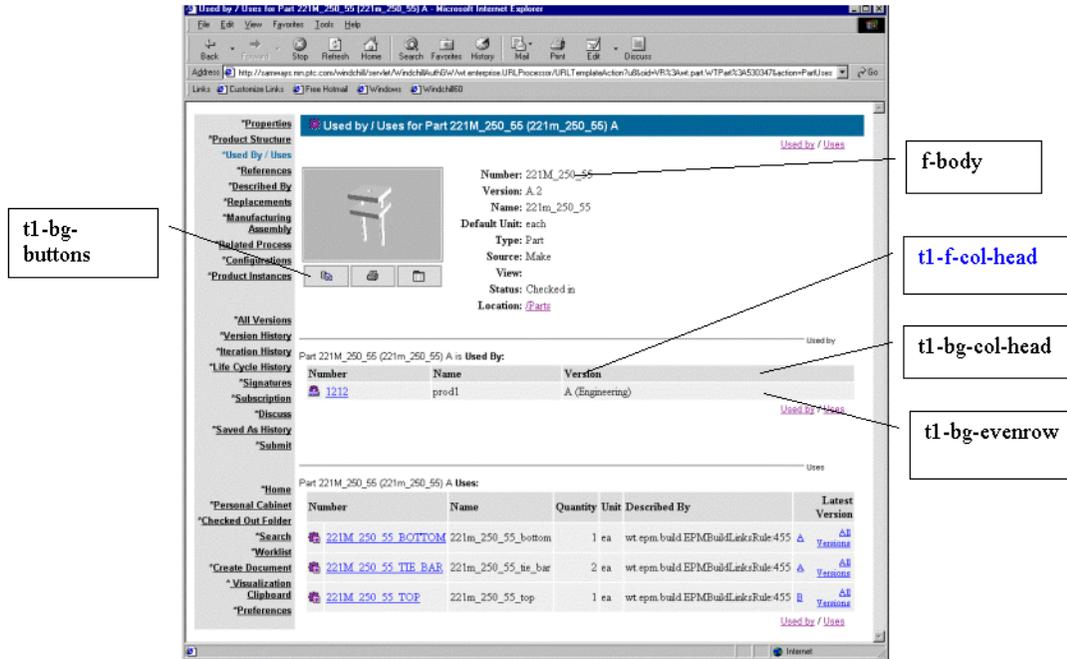
The figures in the remainder of this section show the style classes that are usually used in HTML and JSP clients.

In the figure that follows, the following style classes are not generally used by existing clients but are available for use in new clients:

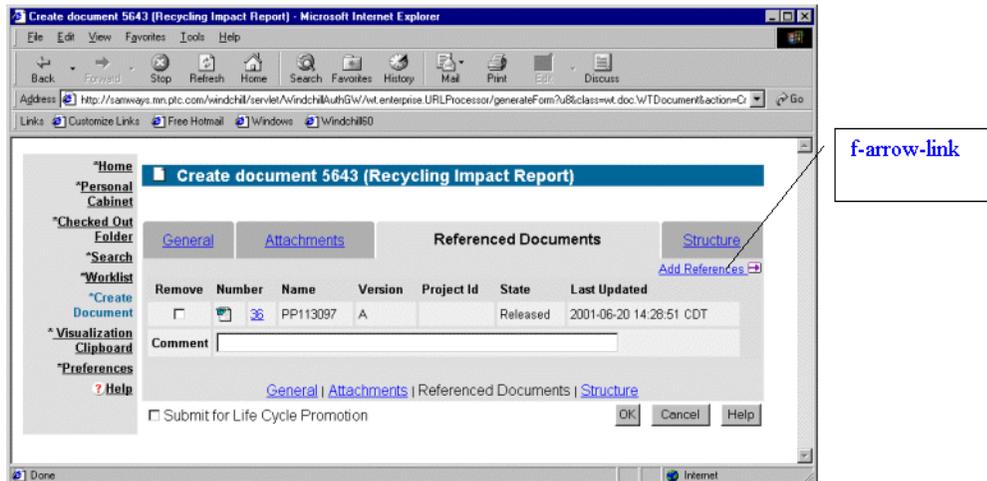
- f-tab-on
- f-tab-off
- f-label
- f-label-sm



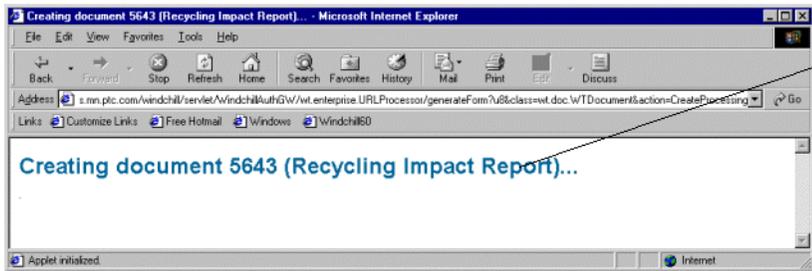
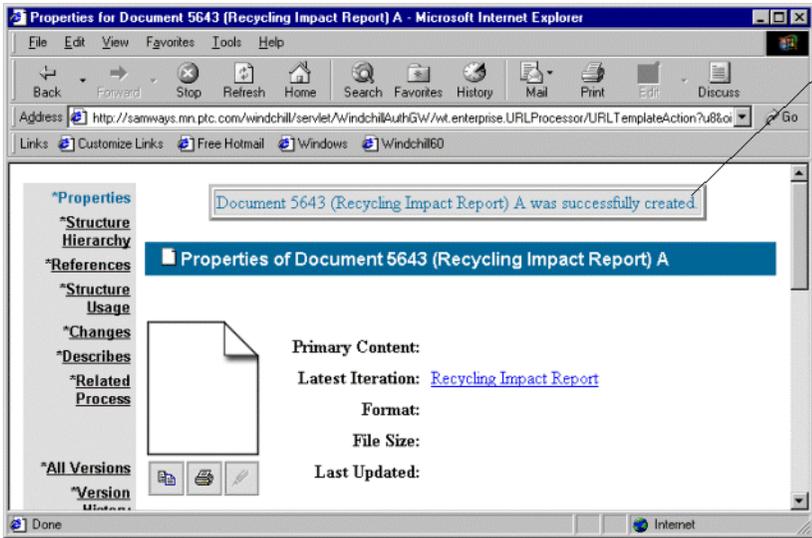
In the following figure, the t1-f-col-head style class is not generally used by existing clients but is available for use in new clients:

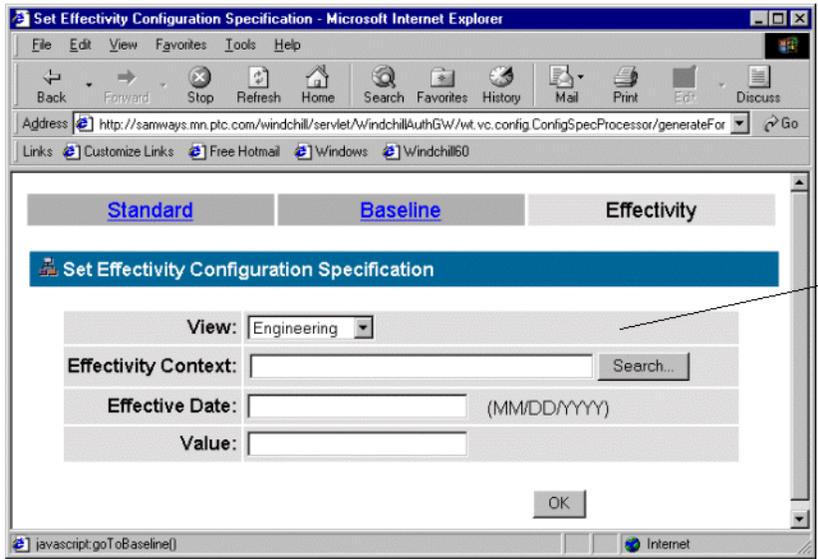


In the following figure, the f-arrow-link style class is not generally used by existing clients but is available for use in new clients:



In the remainder of these figures, the style classes are generally used in existing clients.





## Customizing Localizable Display Names

A display name for a modeled element (class, attribute, or association role) can be customized. These display names are obtained through the PropertyDescriptors that are created by the wt.introspection package. The default display names are defined in the delivered ResourceInfo (.rbInfo) files. The default values can be overridden by placing entries in the ResourceInfo customizations file for modeled packages, which exists in a parallel directory structure. This segregates customizations so that they need not be manually reapplied at each release. The entries placed in a ResourceInfo's customization file override the entries delivered by the package owner.

To change the display name for a modeled element, perform the following steps:

1. Determine the entry necessary to accomplish the customization.
2. Add the customization entry to the appropriate ResourceInfo customizations file.
3. Build the runtime resource bundles for the customized package.
4. Verify the customization.
5. Restart the method servers if they were running during the procedure.

The following example changes the display names for the wt.part.WTPart class and its **locker** attribute. The following steps describe how to determine which ResourceInfo customizations files entries will contain the new display names, and how to set the values. The customization will be made in a location parallel with the ResourceInfo files. The default location for these customizations is \$(wt.home)\wtCustom, as defined by the wt.generation.custom.dir entry in tools.properties. Create this directory if it does not already exist.

1. Determine the entries necessary to accomplish the customizations.
  - a. Obtain an info report for the class by executing the following command:

```
infoReport wt.part.WTPart
```

- b. Inspect the current getDisplayName() values for the WTPart class and its locker attribute:

```
getDisplayName()      : Part
getDisplayName()      : Locked By
```

- c. Inspect the value of the WTIntrospector.DEFINED\_AS property of the **locker** PropertyDescriptor:

```
getValue( WTIntrospector.DEFINED_AS ) : wt.locks.Lock.locker
```

d. Based on this information, use the following values:

Model Element	Customization File	Entry Key
WTPart	Windchill\wtCustom\wt\part\partModelRB.rbInfo	WTPart.Value
locker	Windchill\wtCustom\wt\locks\locksModelRB.rbInfo	Lock.locker.value

2. Add the customization entries to the appropriate ResourceInfo customizations files:

a. Add the following entry to Windchill\wtCustom\wt\part\partModelRB.rbInfo (create this file if it does not exist):

```
WTPart.value=Windchill Part
```

b. Add the following entry to Windchill\wtCustom\wt\locks\locksModelRB.rbInfo (create this file if it does not exist):

```
Lock.locker.value=Lock Held By
```

3. Build the runtime resource bundles for the customized packages by entering the following commands:

```
ResourceBuild wt.part.partModelRB
```

```
ResourceBuild wt.locks.locksModelRB
```

4. Verify the customizations:

a. Obtain an info report for the class and inspect the `getDisplayName()` values as described in the preceding steps. The values should reflect the customizations.

b. If the info report values are unchanged, perform the following steps:

i. Verify that the build step actually updated the following serialized resource bundle file:

```
Windchill\codebase\wt\part\partModelRB.RB.ser
```

ii. Verify the contents of the updated serialized resource bundle:

```
java wt.util.resource.ResourceBundleUtil wt.part.partModelRB
```

To support other locales, make locale-specific copies of the files and add the localized values, as in step 2 above. For further information on localizing resource entries, see the chapter on Internationalization and Localization in the *Windchill Application Developer's Guide*.

## Customizing Local Searches

Windchill provides two types of local search tasks: Search Local and Chooser.

The first type, Search Local, is available from the Windchill Explorer and Product Information Explorer from the task button that looks like binoculars. From the Windchill Explorer, it is also available from the **Tools > Search in Folders** menu. The Search Local task has a class picker that allows selection of a class to search against.

The second type of local search, a Chooser task, is used to choose an object to be used in another screen. An example of this task is the **Find Part** dialog box that appears when the Product Information Explorer is initiated. The class for the search task is preset for the specific task. For example, the **Find Part** dialog box is preset to only query on wt.part.WTPart. After an instance of the class is selected from the search results section of the Chooser, clicking **OK** returns the selected instance to the client that called the Chooser.

Local search tasks can be customized in the following ways:

- Adding your site classes to those that can be used by the Search Local and Chooser tasks.
- Embedding the Search Local task in your own panel.
- Invoking the Chooser task from your own code.
- Changing the date formats used to parse dates for date and time search criteria fields.

The following sections explain background information and give examples on how to customize the local search tasks.

### Adding Your Site Classes

The Search Local and Chooser tasks use files to configure what attributes and classes are available. The Search Local task is configured using wt.clients.query.QueryPanelOptions.java (in source code provided by Windchill). The Chooser task uses the wt.clients.beans.query.ChooserOptions.java (also in source code provided by Windchill).

The following is an excerpt from the ChooserOptions.java file:

```
private static String[][]contents = {

    {"UsrSCM", "C:wt.org.WTUser; G:Search Criteria; A:name;
      A:fullName; " +
      "A:authenticationName"},
    {"GrpSCM", "C:wt.org.WTGroup; G:Search Criteria;
      A:name; " +
      "A:description; G:Advance Criteria;
      A:adminDomainName; " +
      "A:modifyTimestamp"},
```

These two files are formatted similar to resource bundles. To customize them, you simply alter the source, recompile, and put them in the codebase. In `ChooserOptions.java`, each record in the contents structure represents a different class that is used in a Chooser task. The first string in the record ("UsrSCM", for example) is used internally as a key value to reference the information. The client code uses the full path of the class when initiating a Chooser task. As such, adding new records or classes to the `ChooserOptions.java` does not have any impact until client code is either added or changed to call out the new class that was added. The format string:

```
"C:wt.org.WTUser; G:Search Criteria; A:name; A:fullName;
  A:authenticationName"
```

is used to determine the class to search against, the tab text, and the attributes that are used and displayed. The format string is broken up into records. A one-character code followed by a colon begins each record; a semicolon and space combination are used as a separator between records.

The one-character codes are as follows:

- C** Specifies a fully qualified class name to query on. This must be the first record in the format string.
- G** Specifies text for the search criteria tabs. All attributes that follow are grouped on that tab until the next G record.
- A** Specifies an attribute used in both the search criteria section and the display area. Attributes are displayed in the order they appear in the string.
- D** Specifies an attribute used in only the display area.

The following is an excerpt from `QueryPanelOptions.java`:

```
private static String[][]contents = {
    {"All Document Types", "C:wt.doc.WTDocument; G:Search
      Criteria; A:number; " +
      "A:name; D:versionIdentity; A:title; A:department;
      G:More Search Criteria; " +
      "A:lifeCycleState; A:modifyTimestamp;
      A:createTimestamp; " +
      "A:description"},
    {"General", "C:wt.doc.General; G:Search Criteria;
      A:number; " +
      "A:name; D:versionIdentity; A:title; A:type;
      G:More Search Criteria; " +
      "A:lifeCycleState; A:modifyTimestamp;
      A:createTimestamp; " +
      "A:description"},
```

The format of `QueryPanelOptions` is the same as `ChooserOptions.java`. It uses the same key and format string with the same one-character codes, although the way it uses the key string is slightly differently. The key strings (such as "All Document Types") are used to populate the class picker at the top of the Search Local task. Adding new records with different key strings will add more class options to the Search Local task. If you extend one of the existing classes, modify `QueryPanelOptions.java` to add your new class. For example, extend `WTDocument` with a `Training Manual` class with the following new attributes: `language` and `training module`. The following record would have to be added to the `contents` array in `QueryPanelOptions.java` to add `Training Manual` to the Search Local Task:

```
{ "Training Manual", " C:wt.doc.TrainingManual;
  G:Search Criteria; A:number; " +
    "A:name; D:versionIdentity; A:title;
    A:trainingModule;
    G:More Search Criteria; " +
    "A:language; A:department; A:lifeCycleState;
    A:modifyTimestamp; A:createTimestamp; " +
    "A:description" },
```

To see the attributes that can be used in local search tasks, refer to the file `customization/localsearch/queryable.xls` or look at the Rose model for the class.

The `queryable.xls` file is an Excel spreadsheet that lists, by class, the attributes that can be used in the options file.

If you look at the Rose model for a class, attributes listed on the object or, in the case of versioned objects, attributes on either the master or iteration are supported. Run an inforeport with the fully qualified class name, for example:

```
inforeport wt.doc.WTDocument
```

The file `doc.WTDocument.out` will be created in the `TEMP` directory. In the report, if an attribute has `getValue( WTIntrospector.PERSISTENT )` of `TRUE` and `getValue( WTIntrospector.QUERY_NAME )` that is not null, then it is supported. Most object references are supported at this release. Examples of object references are: `modified by`, `project`, and `life cycle template`.

If an attribute is not queryable, it is displayable. The file mentioned earlier, `queryable.xls`, shows which attributes are queryable and which are displayable only in the results.

## Calling the Search Local Task from Client Code

The Search Local task is started by creating an instance of the `QueryPanel` class. You must add listeners to listen for the **Close** button being clicked on the `QueryPanel` or the close on the frame. See the related javadoc in `wt.clients.query`. The following section of code illustrates how to add the Search Local task to a frame.

```
class QueryListener implements QueryPanelListener {
    private Frame parent_frame = null;
```

```

public QueryListener( Frame frame ) {
    parent_frame = frame;
}

public Frame getParentFrame() {
    return parent_frame;
}

public void queryPanelActionPerformed( QueryPanelEvent evt ) {
    if( evt.getAction() == QueryPanelEvent.CANCEL ) {
        parent_frame.dispose();
// Dispose of the frame on a "Close"
    }
}
}

class QueryFrameWindowListener extends java.awt.event.WindowAdapter
{

    public void windowClosing(java.awt.event.WindowEvent event) {
        Object object = event.getSource();
        if( object instanceof Frame ) {
            ((Frame)object).dispose();
        }
    }
}

private void searchLocal() {
    QueryPanel panel = new QueryPanel( getParentApplet() );
// Create the QueryPanel.
    Frame query_frame = new Frame("Local Search");
// Create the frame to put the panel in.
    query_frame.addWindowListener( new QueryFrameWindowListener()
);
// Add a listener to the frame.

    panel.setParentFrame( query_frame );
// Set the parent frame in

// QueryPanel.
    panel.addQueryPanelListener( new QueryListener( query_frame )
);
// Add a listener for the panel.
    query_frame.setLayout( new BorderLayout() );
    query_frame.add( panel, "Center" );
    query_frame.setSize( panel.getSize() );
    query_frame.setLocation( getLocation().x + 30,
        getLocation().y + 30 );
    query_frame.show();
}

```

For examples of how to use the Chooser task, see the *Windchill Application Developer's Guide* chapter on Developing Client Logic, the section on the WtChooser bean.

## Changing Date Formats for Search Criteria

The date formats used for entering dates on search forms are separate from the display formats. If you want to change the date formats for local searches, edit the formats in the `wt.query.dateHelperResource.java` resource bundle. The date formats follow the `java.text.SimpleDateFormat` method of specifying date formats. Recompile the file and replace it in your codebase.

The string-to-date conversion code uses the formats in the file to convert the string into a date that can be used to query against the database. The conversion code attempts to convert the date using first the most specific formats and moving to the least specific formats. More than one format can be acceptable for each level of formats. For example, the search could search for everything created 1998/6/22 or 6/22/1998 if both formats are in the file. Specifying many formats for each level may cause each format to be tried before the correct format is found.

Specify only formats with four digits for the year. Using two digits for the year can cause bad date conversion.

## Customizing the HTML Search

To customize the HTML search to either change the display of the default search objects or to add new classes, see the following file that is distributed as source `Windchill\src\wt\query\SearchAttributeList.java`. As explained in the javadoc for this class, subclass `SearchAttributeList` and make the appropriate entries in `service.properties` and `wt.properties`. Following are methods that should be implemented in a custom `SearchAttributeList`, with examples:

```
public final class MySearchAttributeList extends SearchAttributeList implements
Externalizable {
    public void setLocale( Locale locale ) {

        // Load in the values for the drop down list for selecting what to search
        against.

        clientLocale = locale;

        // **Customize -----
        -----
        //      Add new classes to search to list below.
        //      Make sure that they are assigned numbers in sequence from 0 to N.
        //      Set dropDownListCount to N+1.

        final int ALL                = 0;
        final int WTPART              = 1;
...
        final int MYCLASS            = 22

        int dropDownListCount = 23;
        // -----
        -----

...
        pickList = new String[classCount];
        pickList[ALL] =
WTMessage.getLocalizedMessage(RESOURCE,queryResource.ALL,null,clientLocale);
        pickList[WTPART] =
WTMessage.getLocalizedMessage(RESOURCE,queryResource.WTPART,null,clientLocale);
...
        pickList[MYCLASS] = WTMessage.getLocalizedMessage(RESOURCE,queryResource.
MYCLASS,null,clientLocale);

        pickValues = new String[classCount];
        pickValues[ALL] = queryResource.ALL;
        pickValues[WTPART] = queryResource.WTPART;
...
        pickValues[MYCLASS] = queryResource.MYCLASS;

        // **Customize You will need a string in here to correspond to each item in
pickList
        // The string is a space separated list of what classes to query
        // against. If you want to query against multiple classes that have a common
parent that
        // has all of the attributes that you are interested in use that one class. If
you want
```

```

    // to query against multiple classes that don't have a good common parent then
you can
    // add them to a list and the search will loop through each class and combine
the results
    // at the end. All classes in one list must only search against COMMON
attributes or
    // attributes with the same name and of the same class! If you add both a
parent and
    // a child class to the list you will get duplicate entries, when the results
are
    // combined duplicate entries are not deleted.
queryClass = new String[classCount];

    queryClass[ALL] =
        "wt.part.WTPart wt.doc.WTDocument wt.change2.WTChangeIssue
wt.change2.WTChangeRequest2 " +
        "wt.change2.WTChangeInvestigation wt.change2.WTAnalysisActivity
wt.change2.WTChangeProposal " +
        "wt.change2.WTChangeOrder2 wt.change2.WTChangeActivity2
wt.csm.businessentity.BusinessEntity " +
        "wt.effectivity.ConfigurationItem wt.epm.EPMDocument " +
        "wt.replication.unit.WTUnit " +
        "wt.part.WTProductConfiguration " +
        "wt.part.WTProductInstance2 "; // Please remember to keep a space at the
end so that conditionally added items work.

...

    queryClass[WTPART] = "wt.part.WTPart";

...

    queryClass[MYCLASS] = "??.?.MyClass";

    // **Customize These are the
    // attributes that can be queried against.
    inputAttributes = new String[classCount];
    inputAttributes[ALL] =
        "number name lifeCycleState projectId cabinet creator modifier
modifyTimestamp";
    inputAttributes[WTPART] =
        "number name view versionIdentifier partType source lifeCycleState projectId
cabinet creator modifier modifyTimestamp";

...

    inputAttributes[MYCLASS] =
        "name modifyTimestamp";

    // **Customize Each individual
    // string must match with the string listed above for the inputAttributes. "0"
stands for no
    // input processing. If an attribute is an enumerated type use "0" and the
code will generate
    // the drop down list. In the first string: projectId is in the fourth
position in inputAttributes
    // so the method to generate the drop down list for it is also in the fourth
position in the
    // string. The "0"s and methods must match in number with the number of
attributes listed

```

```

        // under inputAttributes. You may add a fully qualified method from your
customization package
        // as long as it is static and returns a vector of strings.
inputProcessing = new String[classCount];
inputProcessing[ALL] =
    "0 0 0 wt.query.LocalSearchProcessor.getProjectList
wt.query.LocalSearchProcessor.getCabinetList 0 0 0";
inputProcessing[WTPART] =
    "0 0 wt.query.LocalSearchProcessor.getViewList 0 0 0 0
wt.query.LocalSearchProcessor.getProjectList
wt.query.LocalSearchProcessor.getCabinetList 0 0 0";
...
inputProcessing[MYCLASS] =
    "0 0";

    // **Customize This is similar in concept to inputAttributes only these are
the attributes
    // that will be displayed in the search results.
outputAttributes = new String[classCount];
outputAttributes[ALL] =
    "number name versionDisplayIdentifier displayType lifeCycleState projectId
modifyTimestamp";
outputAttributes[WTPART] =
    "number name versionDisplayIdentifier projectId lifeCycleState
modifyTimestamp";
...
outputProcessing[MYCLASS] =
    "ObjProps 0";

    // **New for 6.0
    // **Customize This is similar in concept to outputAttributes only this list
is used
    // to indicate which attributes can be sorted, can't be sorted, or an alternate
attribute
    // that can be sorted to have the same affect as the display attribute. The
string that is used
    // here should be the column descriptor so that it can be used to create the
ClassAttribute for
    // the query. The query that is used for search is a simple query that will
not sort on all
    // of the display attributes. Changing the 0 to 1 for an unsupported attribute
will
    // either cause exceptions or sorts that don't work. Attributes of the
following types are
    // just some examples of the attributes that will either throw exceptions or
sort incorrectly:
    // EnumeratedType, CabinetReference, DataFormatReference,
LifeCycleTemplateReference, ProjectReference,
    // and ViewReference.
sortAttributes = new String[classCount];
sortAttributes[ALL] =
    "1 1 versionInfo.identifier.versionId 0 0 0 1";
sortAttributes[WTPART] =
    "1 1 versionInfo.identifier.versionId 0 0 1";
...
sortAttributes[MYCLASS] =

```

```

        "1 1";

        // **New for 6.0
        // **Customize This is similar in concept to outputAttributes only this list
is used
        // for assigning a unique key to the sort preferences for this search. This
string will
        // be persisted and used to retrieve the sort preferences for users. If the
value of one
        // of these strings is changed or deleted after the system is in operation it
will create orphaned
        // preferences in the system and users will lose the value that they had
persisted for that
        // search. New entries can be added when a new search is added so that sort
preferences
        // can be saved for that new search. These strings are arbitrary and never
displayed to the user.
        sortPref = new String[classCount];
        sortPref[ALL] =
            "all";
        sortPref[WTPART] =
            "wtpart";
...
        sortPref[MYCLASS] =
            "myclass";
    }

    /**
     *
     * <BR><BR><B>Supported API: </B>false
     *
     * @param    locale
     * @return    MySearchAttributeList
     */

    public MySearchAttributeList( Locale locale ) {
        setLocale(locale);
    }

    /**
     *
     * <BR><BR><B>Supported API: </B>false
     *
     * @return    MySearchAttributeList
     */

    public MySearchAttributeList() {

        return;
    }
}

```

wt.query.SearchAttributeList is always the most up to date and should be used as a reference.

The remainder of this section describes two new arrays in wt.query.SearchAttributeList: sortAttributes and sortPref.

Due to the data structures used on some classes, not all attributes that can be displayed in search results are sortable in the search results. The sortAttributes array in wt.query.SearchAttributeList is used to designate which attributes are sortable, and if an alternate attribute should be used for sorting. The version attribute is an example of an alternate attribute used for sorting: the attribute used to display is versionDisplayIdentifier, but the attribute used to sort on is versionInfo.identifier.versionId. Base java types, such as String and int, are sortable. Use the examples in wt.query.SearchAttributeList to determine if any custom types are sortable. Otherwise, a simple test will show if the attribute works, has no effect, or throws an exception.

The sortPref array (shown in the preceding code) is used to define a sort preference base name so users can define their sort preferences for that "Search On" item. A default for the sort preferences should be defined at the system level so that the first time the user uses the system, or if a user never defines preferences, the columns will be sorted logically. A default can be defined using wt.load.LoadFromFile or by using the Preference Administrator editor from the System Administrator portal page

If this is a new database for Release 6.0, the defaults are loaded as part of running the required section of wt.load.Demo (which runs wt.load.LoadFromFile). The site defaults can easily be added to or modified using the Preference Administrator. If the database was created on a system before Release 6.0, wt.load.LoadFromFile can be used to load the base defaults for the delivered configuration of the HTML search classes. See the "PrefEntry.../wt/query/htmlsearch" entries in Windchill\loadFiles\preferences.txt as examples.

Each user preference has an internal name, which is never seen from the client except in the Preference Administrator. Because the current search uses the wt.query.SearchAttributeList to allow users to add new searches, and because there has to be a set of sort preferences for each, a unique sort name is needed for each name in the "Search On" list. Each item in the "Search On" list is not necessarily one object, but can be a list of objects. The sortPref array in wt.query.SearchAttributeList defines a unique string that forms part of the name of the preference. The preferences for sorting are stored in the /wt/query/htmlsearch preference node, and the naming format is as follows:

*<sort preference base name>***sortAttrib***<#>*

*<sort preference base name>***sortDirect***<#>*

The *<sort preference base name>* is the unique string from the sortPref array in wt.query.SearchAttributeList; it has only to be unique within the sort names. The sortAttrib is for the attribute name, and the sortDirect is to indicate ascending or descending. It is false for ascending and true for descending. The *<#>* is the number of the sort key, 0 = first key, and so on. Following are the preferences that are loaded using wt.load.LoadFromFile and Windchill\loadFiles\preferences.txt for the All sort:

```
# All
PrefEntry~allsortAttrib0~number~/wt/query/htmlsearch
PrefEntry~allsortDirect0~false~/wt/query/htmlsearch
PrefEntry~allsortAttrib1~versionInfo.identifier.versionId~/wt/query/htmlsearch
PrefEntry~allsortDirect1~true~/wt/query/htmlsearch
```

In the all-default example, the results are sorted first by the number column and then by the version column, with the number being in ascending order and the version in descending order. Currently, the supported number of sort keys is 3, although theoretically the number of sort keys is limited only by Oracle performance. No testing beyond 3 keys has been done on the system.

# Customizing the Windchill Explorer

This section describes how to add business classes to the Windchill Explorer and how to customize the class of objects that can be created in the Windchill Explorer

## Adding Business Classes to Windchill Explorer

This section describes how to make new business classes available in the Windchill Explorer. For additional information about using Windchill Explorer, see the *Windchill Application Developer's Guide* chapter on Developing Client Logic, the section on the WTE Explorer bean.

The Windchill Explorer supports navigating the hierarchy of Cabinets and SubFolders<sup>1</sup>. Provided the user has proper access rights, any object that implements the `wt.folder.FolderEntry` interface is visible in the Windchill Explorer. That is, when the contents of a Cabinet or SubFolder are retrieved, all `FolderEntry` objects contained in that particular Cabinet or SubFolder are displayed. Consequently, if a new business class is created that implements the `FolderEntry` interface, no additional work must be done for objects of that class to be visible in the Windchill Explorer.

Similarly, if a new business class is created to implement the `FolderEntry` interface, no additional work must be done to make that particular class available in the **File > New** menu of objects to create in the Windchill Explorer. (For further information, see the following section on Customizing objects created from the Windchill Explorer.) The list of classes displayed when the user chooses to create a new object is retrieved by finding all concrete classes implementing the `FolderEntry` interface. Consequently, a new business class that implements the `FolderEntry` interface is immediately available in the list of creatable classes. However, while the new class is available in this list, there is no corresponding logic to perform the actual creation.

To be able to create, update, view, and delete new business classes implementing the `FolderEntry` interface from the Windchill Explorer, additional work is required. To provide this support for creating, updating, viewing, and deleting `FolderEntry` objects, the Windchill Explorer makes use of task delegates (`wt.clients.util.TaskDelegate`). Being able to create, update, view, or delete a `FolderEntry` object from the Windchill Explorer requires only having an implementation of a `TaskDelegate` for that particular object.

---

1. From an implementation perspective, a Folder (`wt.folder.Folder`) is an interface implemented by objects that act as containers of information. In the Windchill system, Cabinets (`wt.folder.Cabinet`) as well as SubFolders (`wt.folder.SubFolder`) implement the `Folder` interface. From a real world perspective, however, many view folders as different from cabinets—both are information containers, but folders exist inside of cabinets, and cabinets are not synonymous with folders. To avoid confusion, this section refers to Windchill Explorer objects by their implementation names (that is, Cabinets and SubFolders).

Consider the following example. Suppose you want to create a new type of business class to provide an implementation for a Report object. The fully qualified name of the class will be `mysite.report.MyReport`. Because the `Foldered` interface implements `FolderEntry`, you can create the new class by defining the `MyReport` class to implement the `Foldered` interface, as follows:

```
public class MyReport extends WObject implements Foldered {
    ...
} // End of MyReport''
```

To be able to create `MyReport` objects, you must build a GUI frame:

```
public class CreateMyReportFrame extends java.awt.Frame {
    ...
    public void setParentFolder( Folder folder ) {
        // Initialize the folder in which to create the
        // new MyReport
    }
    ...
} // End of CreateMyReportFrame''
```

Similarly, you must build GUIs for updating and viewing `MyReport` objects:

```
public class UpdateMyReportFrame extends java.awt.Frame {
    ...
    public void setMyReport( MyReport report ) {
        // Set the MyReport to be updated
    }
    ...
} // End of UpdateMyReportFrame''

public class ViewMyReportFrame extends java.awt.Frame {
    ...
    public void setMyReport( MyReport report ) {
        // Set the MyReport to be viewed
    }
    ...
} // End of ViewMyReportFrame''
```

After you build the GUIs to create, update, and view `MyReport` objects, you must integrate these GUIs into the Windchill Explorer. This integration is done by implementing a `TaskDelegate`.

All create, update, view, and delete tasks are launched through `TaskDelegates` in the Windchill Explorer. When an update is invoked on an object, for example, the Windchill Explorer attempts to locate a `TaskDelegate` implementation based on the class of the selected object. If a `TaskDelegate` is found, the Windchill Explorer invokes the `launchUpdateTask` method on that `TaskDelegate`.

The TaskDelegate implementations must conform to a particular naming convention in order for the Windchill Explorer to find them. If a given Foldered object has a fully qualified name of xxx.yyy.MyObject, the fully qualified name of the corresponding TaskDelegate implementation must be xxx.clients.yyy.MyObjectTaskDelegate. So, for example, the TaskDelegate implementation corresponding to wt.doc.WTDocument is wt.clients.doc.WTDocumentTaskDelegate.

If a TaskDelegate cannot be found corresponding to the fully qualified classname, the same search algorithm will be repeated on the parent class of the given class. If the parent class results in no TaskDelegate being found, the same search algorithm will be repeated on the interfaces the given class implements.

The TaskDelegate implementation for the MyReport class would have a fully qualified name of mysite.clients.report.MyReportTaskDelegate. To implement each of the tasks to create, update, view, and delete MyReport objects, you launch the respective frames:

```
package mysite.clients.report;

public class MyReportTaskDelegate extends
    wt.clients.util.TaskDelegate {

    public void launchCreateTask() throws
        TaskDelegateException {
        CreateMyReportFrame create_frame = new
            CreateMyReportFrame();
        create_frame.setParentFolder( getParentFolder() );
        create_frame.show();
    } // End of 'launchCreateTask'

    public void launchUpdateTask() throws
        ModificationNotAllowedException,
        NotAuthorizedException,
        ObjectNoLongerExistsException,
        TaskDelegateException,
        WTException {

        if( getObject() == null ) {
            throw new TaskDelegateException("No MyReport
                to Update");
        }

        if( !(getObject() instanceof MyReport) ) {
            throw new TaskDelegateException("Object is not
                of type MyReport");
        }

        // Refresh the object before launching the update
        super.launchUpdateTask();

        UpdateMyReportFrame update_frame = new
            UpdateMyReportFrame();
        update_frame.setMyReport( (MyReport)getObject() );
        update_frame.show();
    }
}
```

```

    } // End of 'launchUpdateTask'

    public void launchViewTask() throws NotAuthorizedException,
        ObjectNoLongerExistsException, WTEException,
        TaskDelegateException {
        if( getObject() == null ) {
            throw new TaskDelegateException( "No MyReport
                object to View");
        }

        if( !(getObject() instanceof MyReport ) ) {
            throw new TaskDelegateException( "Object is not of
                type MyReport" );
        }

        // Invoke method already defined in super class to
        // display the HTML properties page for this object.
        displayPropertiesPage();
    } // End of 'launchViewTask'

    public void launchDeleteTask() throws
        TaskDelegateException, WTEException {
        if( object instanceof Persistable ) {

            String identity = "";
            if( object instanceof WXObject ) {
                LocalizableMessage localized_message =
                    ((WXObject)object).getDisplayIdentity();

                identity = localized_message.getLocalizedMessage(
                    WTXContext.getContext().getLocale() );
            } else {
                identity = object.toString();
            }

            Object[] params = { identity };
            String message = WTMessage.getLocalizedMessage(
                RESOURCES,
                UtilRB.CONFIRM_DELETE_OBJECT,
                params );
            if( confirmAction( message ) ) {
                deleteObject();
            }
        }

        } // End of 'launchDeleteTask'

    } // End of 'MyReportTaskDelegate'

```

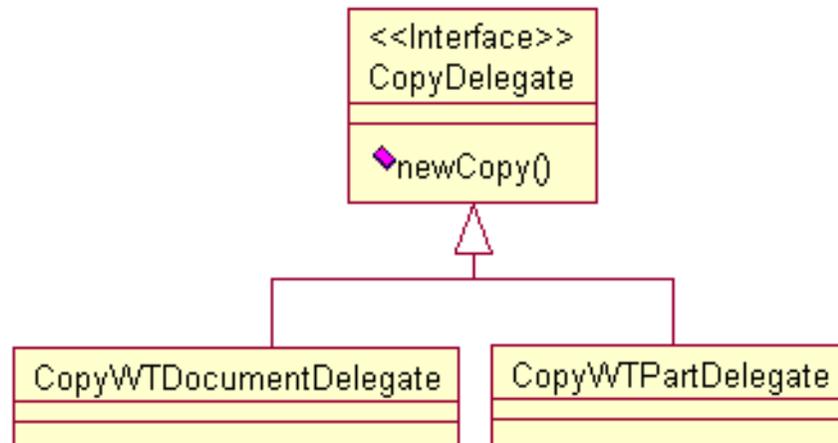
**Note:** For simplicity in the preceding example, hard-coded strings were used to initialize the TaskDelegateExceptions. We recommend using resource bundles in the actual implementation.

Once the TaskDelegate for the new business class is in place, the create, update, and view actions for the new business class will be available from the Windchill Explorer.

## Adding 'Save As' Functionality to a New Document or Part Class

To allow a new business class that is a child of either wt.doc.WTDocument or wt.part.WTPart to use the Save As action, you must create a copy delegate that knows how to create a copy of the object and copy the attributes of the original object to the new object. If a copy delegate is not created, the Save As action creates a WTDocument or WTPart instance for the copy.

The simplest way to create the delegate is to model a subclass of wt.enterprise.CopyDelegate the same way that CopyWTDocumentDelegate is modeled. Do not model the new copy delegate as a subclass of CopyWTDocumentDelegate or CopyWTPartDelegate because then the code generation does not generate the newCopy method.



First, generate your new class, mypackage.CopyMyDocumentDelegate. Add code similar to the following in the generated file:

```
/**begin newCopy%378E0B7F037Af.doc preserve=no
/**
 * @param    object
 * @return    RevisionControlled
 * @exception wt.util.WTException
 **/
/**end newCopy%378E0B7F037Af.doc

public RevisionControlled newCopy( RevisionControlled object )
    throws WTException {

    if (object == null) {
        throw new WTException(RESOURCE,
            wt.enterprise.enterpriseResource.COPY_NULL_OBJECT,
            null);
    }
}
```

```

    }
    if (!(object instanceof MyDocument)) {
        Object[] params = {object.getConceptualClassname()};
        throw new WTEException(RESOURCE,
            wt.enterprise.enterpriseResource.COPY_BAD_CLASS,
            params);
    }

    try {
        MyDocument new_doc = MyDocument.new MyDocument ();
        MyDocument orig = (MyDocument) object;
        new_doc.setMyAttribute(orig.getMyAttribute());
        new_doc.setTitle(orig.getTitle());
        new_doc.setDescription(orig.getDescription());
        new_doc.setDepartment(orig.getDepartment());
        new_doc.setDocType(orig.getDocType());
        return new_doc;
    }
    catch (WTPropertyVetoException wtpve) {
        throw new WTEException(wtpve, RESOURCE,
            wt.enterprise.enterpriseResource.COPY_ATTRIBUTES,
            null);
    }
    //##end newCopy%378E0B7F037Af.body
}

```

Next, add any custom attributes that must be copied, similar to myAttribute in the preceding example.

Finally, add the following line to wt/enterprise/EnterpriseServerDelegate.properties in the codebase directory:

```

wt.services/svc/default/wt.enterprise.CopyDelegate/null/
  mypackage.MyDocument/0=mypackage.CopyMyDocumentDelegate/
  duplicate

```

Note you should actually enter only one line; the indentation here indicates a continuation of the preceding line, necessary for presentation of lines within the book.

## Modifying the Copy Rules for 'Save As' Functionality

Part of the Save As functionality uses rules in the wt.properties file to configure how the functionality is executed. When adding the Save As functionality to a custom class, you can add to these rules to change the behavior of Save As on that custom class. The custom class must be a RevisionControlled object for the Save As to work.

The first rule (shown below) specifies the delimiter used in the rules. You should not change it unless absolutely necessary and, in that case, you must change the delimiter for all of the rules.

```

# Copy rules for Save As functionality.
wt.enterprise.copyRuleDelimiter=,

```

The second rule, the copy rules, has the following format:

*<property key n>=<class name>,<rule type>,<value>*

The property key is used to retrieve the rules from the property file. Copy rules are numbered sequentially, starting with 0. Do not skip numbers. You can add additional rules and should start numbering at the last number plus 1.

The class name is the fully qualified path of the class. The rules use a very simplistic resolution of the class hierarchy: when a rule that matches the class is found, the parents of that class are not searched for any further rules. Therefore, if a rule is missing at that level of the class, the default is used. It is better to repeat all of the rules for a class, even if only one custom rule is required for a class.

The rule type is one of the three types of copy rules: Content, MadeFrom, and Relationship.

### **Content**

Content is a boolean rule that defines if content should be copied when a Save As action is performed. This is the default if there is no rule to copy content.

### **MadeFrom**

MadeFrom is a boolean rule that defines if the MadeFrom relationship should be created. It is the default if there is no rule to create the relationship.

### **Relationship**

Relationship rules define which BinaryLink relationships should be copied. Relationship rules define the relationship and the role that should be copied. The default, if no relationship rules are present, is to copy no relationships.

The following is an excerpt of the rules from wt.properties:

```
wt.enterprise.copyRules0=wt.doc.WTDocument,Content,true
wt.enterprise.copyRules1=wt.doc.WTDocument,MadeFrom,true
wt.enterprise.copyRules2=wt.part.WTPart,Content,true
wt.enterprise.copyRules3=wt.part.WTPart,MadeFrom,true
wt.enterprise.copyRules4=wt.part.WTPart,Relationship,
wt.part.WTPartUsageLink-uses
wt.enterprise.copyRules5=wt.part.WTPart,Relationship,
wt.part.WTPartReferenceLink-references
wt.enterprise.copyRules6=wt.part.WTPart,Relationship,
wt.part.WTPartDescribeLink-describedBy
. . .
wt.enterprise.copyRules18=wt.doc.WTDocument,Relationship,
wt.doc.WTDocumentUsageLink-uses
wt.enterprise.copyRules19=wt.doc.WTDocument,Relationship,
wt.doc.WTDocumentDependencyLink-dependsOn
```

The third rule is the copy service rules. After a relationship has been added into the copy rules as described above, there is the option of the copy trying to copy it during the Save As action or allowing the service associated with the link class to copy it by listening for events. The copy code uses `wt.enterprise.CopyBinaryLinkDelegate` to copy relationships. `CopyBinaryLinkDelegate` currently supports copying Object to Object, Object to Version, and Version to Version links. Link classes that are not one of these link types should not have an entry in the copy service rules and should have the owning service copy the relationships. If the owning service of the relationship is doing the copy, there is no need to have an entry for this relationship in the copy rules either, because the copy rule is not checked by the owning service. The following is an abridged listing of the copy service rules. New links can be added to the end of the rule by adding a comma (,) and then the fully qualified link class name.

```
wt.enterprise.copyServiceRules=wt.part.WTPartUsageLink,  
wt.part.WTPartReferenceLink,wt.doc.WTDocumentUsageLink,  
wt.doc.WTDocumentDependencyLink
```

## Customizing Objects Created from the Windchill Explorer

Classes that implement the `wt.folder.FolderEntry` interface will appear as an option in the list of classes that can be created from the Windchill Explorer. That is, a class that implements the `wt.folder.FolderEntry` interface will appear in the **File > New** menu and in the list of classes displayed in the dialog launched when clicking the **New** toolbar button. It is possible to add additional classes to and remove existing classes from the list of classes that can be created from the Windchill Explorer.

Currently, the Windchill Explorer builds the list of classes that can be created by finding all concrete classes implementing the `wt.folder.FolderEntry` interface. After finding all concrete subclasses, the Windchill Explorer uses the `FolderExplorerMenuRB` resource bundle to further customize the classes available. The `FolderExplorerMenuRB` resource bundle is generated from the `wt/clients/folderexplorer/FolderExplorerMenuRB.rbInfo` file. The `FolderExplorerMenuRB.rbInfo` file contains the definition of a key, `NEW_MENU_OBJECTS`, whose value contains information that is used in customizing the class of objects that can be created in the Windchill Explorer. Entries corresponding to this key have the following form:

```
<operator><class name>[:<display name>];
```

where *operator* is either '+' (to add a class) or '-' (to remove a class); *class name* is the fully qualified class name of the class to be added or removed; and *display name* is an optional name to be displayed in the menu for that class. The semi-colon (;) separates the entries.

Consider the following example:

```
0.constant=NEW_MENU_OBJECTS
0.value=-wt.change2.WTChangeActivity2;-wt.folder.ShortcutLink;
-wt.folder.IteratedShortcutLink;+wt.folder.Shortcut:Link;
```

This entry will remove `WTChangeActivity2`, `ShortcutLink`, and `IteratedShortcutLink` and add `Shortcut` to the class of objects that can be created from the Windchill Explorer. The entry for `Shortcut` will have a label of "Link".

Note that while this mechanism provides some support for customizing the class of objects that can be created in the Windchill Explorer, this API may change in future releases to provide more support for customization.

## Making Parts Content Holders

By default, parts are not content holders. The functionality exists to make them so; however, PTC recommends that you not do so unless absolutely necessary.

To turn on the ContentHolder bean for WTPart in the Java client, set the wt.part.showContents property in the wt.properties file to true.

To turn on the Content link in the navigation bar for part HTML pages, edit the file wt\enterprise\UrlLinkResource.java to add the following string to the value WTPART\_ACTIONS:

```
Content:Contents
```

The WTPART\_ACTIONS value should then look similar to the following:

```
{ WTPART_ACTIONS,
  "ObjProps:Properties,part_structure:Product Structure,
  PartUses:Uses,PartUsedBy:Used By,
  Content:Contents,PartReferences:References,
  BuildRuleBuildSource:Described By",
  "AssociatedProcess:Related Process"
```

## Adding URLs to the Method Server

### Writing Methods That Handle HTTP Requests

The HTTP gateway expects the signature of (HttpServletRequest req, HttpServletResponse resp) for methods handling HTTP GET requests. The signature of methods handling HTTP POST requests is (HttpServletRequest req, HttpServletResponse resp, InputStream is). Note that in processing a POST request, the HTTP gateway will call the Post method to do the post, and then call the Get method of the same name to produce any HTML output. The output stream is available only to the Get method. An example follows:

```
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import wt.util.MPInputStream;
import java.util.Properties;
import java.util.Vector;

import wt.httpgw.CGIConstants;
import wt.httpgw.HttpServletRequest;
import wt.httpgw.HttpServletResponse;
import wt.htmlutil.ProcessTemplate;
import wt.htmlutil.HtmlUtil;
import wt.htmlutil.HTMLTemplate;
import wt.htmlutil.TemplateName;
```

```

import wt.httpgw.LanguagePreference;
import wt.util.WTException;

/**
 * This is an example class containing methods to process
 * two different
 * HTTP Post requests and a method to process an
 * HTTP Get request. This class extends HTMLUtil so that
 * the static methods
 * contained within HTMLUtil do not have to
 * be referenced by
 * HTMLUtil.methodname. This code is for illustration purposes
 * and comes with no guarantee.
 */
public class Example extends HtmlUtil implements CGIConstants{
    private static String name = "default";

    // The first example of a POST request requires a 'post'
    // method be
    // present.
    // This is not always required. If the post is a simple
    // post with
    // URL encoded
    // data, the data manipulation can be done in the 'get'
    // method which
    // is called
    // directly if no corresponding 'post' method exists.

    // This example presumes input from the request input
    // stream is a file
    // being uploaded via a POST request from a multipart
    // form-data on
    // the client. It utilizes the MPInputStream filter class
    // to parse out the
    // multipart form and save the file to the local directory
    // specified in the value of the POST_DIR query string parm.
    // This is all
    // accomplished in the readPost(HttpServletRequest, HttpServletResponse,
    // InputStream)
    // method. Once this is complete, the
    // readPost(HttpServletRequest, HttpServletResponse)
    // version of the method is called. This version of the
    // method sets appropriate
    // response headers and writes its response to the response's
    // output stream.

/**
 * This version of the readPost method handles post requests
 * with object bodies.
 *
 * @param req HttpServletRequest object we received.
 * @param res HttpServletResponse object we are sending back.
 * @param is InputStream object we received to read from
 * (actually req.input_stream).
 *
 */
}

```

```

public static void readPost (HttpServletRequest req, HttpServletResponse res,
    InputStream is)
    throws WTEException
{
    Properties query_string = null;
    String post_dir = null;

    query_string = req.splitQueryString(
        req.getProperty(CGI_QUERY_STRING));
    post_dir = query_string.getProperty("POST_DIR", ".");

    int x = 0;
    byte buffer[] = new byte[1024];
    String fname = new String();
    MultipartInputStream mis = new MultipartInputStream(req.getInputStream(),
        req.getProperty(CGI_MULTIPART_BOUNDARY).toString() );
    try {
        while (mis.hasMoreObjectBodies() ) {
            if ((fname = mis.getBodyHeader("filename")) != null) {
                // replace doublequote with space.
                fname = fname.replace('\u0022', '\u0020');
                fname = fname.trim();
                // need to check if and what kind of dir delimiter.
                // client could be different than host system.
                int u = fname.lastIndexOf("/");
                int d = fname.lastIndexOf("\\");
                if ( u <= d ) {
                    fname = fname.substring(fname.lastIndexOf("\\")
                        +1);
                } else if ( u > d ) {
                    fname = fname.substring(fname.lastIndexOf("/")
                        +1);
                }
                // If both u and d where zero, we had no path.
                // This example doesn't care about path where

                // file came from
                // but other production business classes may&hellip;
                FileOutputStream OutFile = new
                    FileOutputStream( post_dir +
                        System.getProperty("file.separator") +
                        fname );
                while ((x = mis.read(buffer, 0, buffer.length))
                    >= 0) {
                    OutFile.write(buffer, 0, x);
                }
                OutFile.flush();
                OutFile.close();
            }
        }
    }
    catch (IOException e) {
        throw new WTEException (e);
    }
}

```

```

/**
 * This version of the readPost method handles the
 * response for the POST.
 * Insure that you set all the HTTP response headers
 * that you want
 * prior to getting the output stream with the
 * response objects
 * getOutputStream call.
*<p>
* @param req HttpServletRequest object we received.
 * @param res HttpServletResponse object we are sending back.
 *
**/
public static void readPost (HttpServletRequest req, HttpServletResponse res)
    throws WTEException
{

    // This could be a post of non-file data,
    // or it could be the second call to this overloaded
    // method after the initial call above processed the
    // inputStream.
    // If the latter, we could set response stuff here, etc.
    res.setStatus(200);
    if (Integer.parseInt(req.getProperty(CGI_CONTENT_LENGTH))
        == 0) {
        res.setStatus(204, "No Content");
    }

    // res.getOutputStream causes the headers to be written out.
    // So, set all headers prior to getting the output stream.
    try {
        PrintWriter out = new PrintWriter(res.getOutputStream());
        out.println (beginHtml ("Thank You"));
        out.println("Thank you for the post!");
        out.println("Content length was
            " + Integer.parseInt(req.getProperty(
                CGI_CONTENT_LENGTH)));
        out.println (endHtml());
        out.flush();
    } catch (IOException io) {
        throw new WTEException (io);
    }

}

// The following example is a 'get' method that processes
// both a GET
// request
// and a simple POST request.  If getting posted data,
// one input
// parameter
// is expected: 'name'.

/**
 * The GET request handling method.  An HTML page is

```

```

* generated which
* returns back the current value of the class variable
* name. Note
* that this is called directly on a POST request because
* no corresponding POST method
* exists and the request contained simple URL encoded data.
*
* @param req      The HTTPRequest structure.
* @param resp     The HTTPResponse structure from which
* we get
* the output stream.
**/
public static void processPost (HTTPRequest req,
    HTTPResponse resp)
    throws WTEException {

try {

        if (req.isPostRequest()) {
            // This Get method follows a
            // Post method call.
            // Get the
            // data set in the Post method.
            // Since no
            // corresponding POST method
            // exists, this method gets the
            // data directly.
            try {
                Properties inputContents =
                    req.getFormData();
                name = inputContents.getProperty ("name");
            } catch (Exception e) {
                throw new WTEException (e);
            }
        }

        PrintWriter out = new PrintWriter
            (new BufferedWriter (
                new OutputStreamWriter(
resp.getOutputStream())));

        String comment = insertComment ("The following was
            generated dynamically by
            calling the htmlutil methods
            from within java code.");
        out.println (comment);

        out.println (beginHtml ("example"));

        out.println (formatLayout ("P", true, null, "The last
            name posted
            is: " + name));
        out.println (endHtml());
        out.flush();
    }
    catch (Exception e) {
        throw new WTEException(e);
    }
}

```

```

}

// The following example is a 'get' method that only
// processes a GET
// request.

/**
 * This is a GET request handling method. An HTML page
 * is generated
 * from a template file.
 * The template name was specified on browser input. There
 * is no corresponding
 * POST method as
 * none are allowed.
 *
 * @param req The HttpServletRequest structure.
 * @param resp The HttpServletResponse structure from which we get the
 * output stream.
 */
public static void processGet (HttpServletRequest req,
    HttpServletResponse resp) throws
    WTEException {

    String queryString =
        req.getProperty(CGI_QUERY_STRING);

    Properties queryStringContents =
        req.splitQueryString(queryString);
    String name = queryStringContents.getProperty
        ("Templatename");

    // Generate the page from a template which
    // contains a
    // Windchill script.

try {
    // Get the preferred language of the template.
    String acceptLanguage =
        req.getProperty(req.CGI_ACCEPT_LANGUAGE);
    Vector preferences =
        LanguagePreference.getAcceptLanguagePreferences(
            acceptLanguage);

    OutputStream os = resp.getOutputStream();

    // Get the full resource name of the template. In this
    // case, it will
    // be /templates/name. Use this
    // when instantiating a new HTMLTemplate object.
    HTMLTemplate template = new HTMLTemplate
        (TemplateName.getWindchill(name));

    // Find and open the template that is the best
    // match for the
    // preferred language(s) given

```

```

        template.init (preferences);

    // Process the template.  The class context specified
    // here is 'null'.
    // Therefore, all methods listed in the
    // template to invoke must be static methods and the
    // class must be
    // specified in the template.  See
    // below for information on HTML templates.
        template.process (os, null);
    }
    catch (IOException io) {
        throw new WTEException (io);
    }
}
}
}

```

As already stated, the output stream is unavailable to methods handling POST requests. However, the response headers array is available, and the headers can be set in the POST request method. Note that any FORM input into the POST method is retrieved from the req.inputStream. Any name/value pairs on the FORM action URL must be retrieved from the CGI\_QUERY\_STRING. See the GET methods for an example. Input can be passed only to the GET methods through the CGI\_QUERY\_STRING.

The URL to call the processPost method as a get request would look similar to the following:

```
http://Windchill/servlet/WindchillGW/Example/processPost
```

## Generating HTML Dynamically

HTML text can be generated dynamically in two ways. The first technique uses template files. The template file consists of HTML text with embedded Windchill script calls. When a method makes use of a template, it instantiates an HTMLTemplate object for that method. (For further information, see Chapter 7, Customizing the HTML Client.) The process method of HTMLTemplate calls the method processTemplate, which parses the HTML text, looking for any embedded Windchill scripts. When a Windchill script is found, the method(s) specified within are invoked and the HTML text produced is inserted in the output stream. The inputs to the method specified within the Windchill script are in name=value pairs. These inputs are then passed to the method in a Properties object. The methods invoked by the template processor must have the following signature:

```
(Properties prop, Locale locale, OutputStream out)
```

An example follows. The template file contains:

```

<HTML>
<HEAD><TITLE>Some Title</TITLE></HEAD>
<BODY>
Some lines may be here, forms, tables etc.

```

```

<!--Now put in a call to a method to dynamically insert HTML text -- >
<!--We put the list of <class><method><parameters> inside a comment
<!--for those
cases where this HTML may->
<!--be directly displayed on the browser. Each
<!--<class><method><parameters> call
should be on a separate line->
<!--Note that the <class> does not have to be present. If only\
<!--the method (and
optionally <parameters> is present->
<!--then when the template processor is called, it is called with a
<!--class context that
contains the method(s) to invoke.->
<SCRIPT LANGUAGE=Windchill>
  <!--
  wt.htmlutil.test.GenerateDriver insertForms
filename=C:\Windchill\codebase\wt\htmlutil\test\TestDriverForms.html
  --->
</SCRIPT>
</BODY>
</HTML>

```

insertForms is a simple method that reads in the file specified and writes it to the output stream. If TestDriverForms.html contains the following HTML text:

```

<FORM method='POST' action=''http://127.0.0.1/Windchill/servlet/WindchillGW/
  wt.htmlutil.test.TestPage/processPage''>
Name:<INPUT type='TEXT' name='name' maxlength=50>
<P><INPUT type='SUBMIT' ALIGN='CENTER'></P>
</FORM>

```

The HTML page received by the browser will read as follows:

```

<HTML>
<HEAD><TITLE>Some Title</TITLE></HEAD>
<BODY>
Some lines may be here, forms, tables, etc.
<!--Now put in a call to a method to dynamically insert HTML text->
<FORM method='POST' action=''http://127.0.0.1/Windchill/servlet/WindchillGW/
  wt.htmlutil.test.TestPage/processPage''>
Name: <INPUT type='TEXT' name='name' maxlength=50>
<P><INPUT type='SUBMIT' ALIGN='CENTER'></P>
</FORM>
</BODY>
</HTML>

```

**Note:** This is a simple example for illustration purposes. The URL given really should be generated through a method in a Windchill Script.

The second way to dynamically generate HTML text is to use the methods found in `wt.htmlutil.HtmlUtil`. To produce the preceding output, given that the output stream is named 'out', the content of the java method will look similar to the following:

```
String str = beginHtml("Some Title") + "Some lines may be here,  
forms, tables etc."  
    str += insertComment ("Now put in a call to a method to  
        dynamically insert  
        HTML text");  
  
    String formContents = "Name:" +addFormInput  
        ("type=\"TEXT\" name=\"name\"  
wt.httpgw.URLFactory url_factory = new URLFactory( );  
str += createForm( wt.httpgw.GatewayServletHelper.buildAnonymousHREF url_factory,  
    "wt.htmlutil.test.TestPage", "processPage" ), "POST", null, null, formContents );  
str += endHTML( );  
  
out.println (str);  
out.flush();
```

The preferred way to dynamically generate HTML is with the HTML templates. It lends itself to easy customization and localization.

# 5

## Customizing service.properties

Topic	Page
Customizing service.properties .....	5-2

## Customizing service.properties

The service delegate mechanism uses Java property files to specify the delegates that are used for each service for a given set of criteria. The main property file is named `service.properties` and is located in `/Windchill/codebase/`. (For further information, see the section on property files in the chapter titled *The Windchill Development Environment* in the *Windchill Application Developer's Guide*.)

Instead of adding new entries to the `service.properties` file, or overriding existing entries in it, use a separate file. This file must have entries with the same format as those in `service.properties`. To use the new property file, add the file's full path (relative to the system classpath) to a comma-separated list of files in the following property located in `wt.properties`:

```
wt.services.applicationcontext.WTServiceProviderFromProperties.  
    customPropertyFiles
```

Consider the example of creating a new `wt.identity.DisplayIdentification` delegate. The `DisplayIdentification` service is an interface that defines methods for creating strings that identify an object for user interface display purposes. In `service.properties`, several entries exist for the `DisplayIdentification` service, as follows:<sup>1</sup>

```
# The wt.identity.DisplayIdentification service.  
#  
# Delegate definitions for all objects  
#####  
wt.services/svc/default/wt.identity.DisplayIdentification/null/  
    java.lang.Object/0=  
    wt.identity.DisplayIdentificationObjectDelegate/  
    duplicate  
wt.services/svc/default/wt.identity.DisplayIdentification/null/  
    wt.fc.Persistable/1=  
    wt.identity.DisplayIdentificationPersistableDelegate/  
    duplicate  
wt.services/svc/default/wt.identity.DisplayIdentification/null/  
    wt.folder.Folder/2=  
    wt.folder.DisplayIdentificationFolderDelegate/  
    duplicate  
wt.services/svc/default/wt.identity.DisplayIdentification/null/  
    wt.folder.Shortcut/2=  
    wt.folder.DisplayIdentificationShortcutDelegate/  
    duplicate  
wt.services/svc/default/wt.identity.DisplayIdentification/null/  
    wt.doc.WTDocumentMaster/2=  
    wt.identity.DisplayIdentificationWTDocumentMasterDelegate/  
    duplicate  
wt.services/svc/default/wt.identity.DisplayIdentification/null/  
    wt.enterprise.RevisionControlled/2=  
    wt.identity.DisplayIdentificationRevisionControlledDelegate/
```

- 
1. Indentation in this example indicates a continuation of the preceding line, necessary for presentation in the manual. The entry for each property in a property file can be on only one line.

```

duplicate
wt.services/svc/default/wt.identity.DisplayIdentification/null/
wt.part.WTPartMaster/2=
wt.identity.DisplayIdentificationWTPartMasterDelegate/
duplicate
wt.services/svc/default/wt.identity.DisplayIdentification/null/
wt.part.WTPart/3=
wt.identity.DisplayIdentificationWTPartDelegate/
duplicate

```

Each entry maps a class or interface, such as `wt.part.WTPart`, to a class that implements the interface `wt.identity.DisplayIdentification`, such as `wt.identity.DisplayIdentificationWTPartDelegate`. If a customization includes a subclass of `WTPart` named `AcmePart`, and that part has different display identification needs than `WTPart`, a new delegate must be implemented. After the delegate class is implemented, it is used only when the correct entry has been added to a service delegate property file. To do so, perform the following steps:

1. Create a new property file named `acme_part_services.properties` to contain all entries for service delegates that apply to the `AcmePart` class.
2. Put this property file in the directory for the appropriate package, in this example, `/Windchill/codebase/com/acme/part`.
3. Add the name of the new property file to the list of files specified in the `wt.properties` file property named `wt.services.applicationcontext.WTServiceProviderFromProperties.customPropertyFiles`, as follows:<sup>2</sup>

```

wt.services.applicationcontext.WTServiceProviderFromProperties.
  customPropertyFiles=com/acme/doc/
  acme_part_services.properties

```

4. Restart the method server in order for the change to take effect.

---

2. Indentation in this example indicates a continuation of the preceding line, necessary for presentation in the manual. The entry for each property in a property file can be on only one line.



# 6

## Creating Load Methods

<b>Topic</b>	<b>Page</b>
Overview of Load Utilities.....	6-2
Customizing Loading .....	6-4
Creating New Methods for Loading.....	6-6
Existing Load Methods .....	6-9

# Overview of Load Utilities

The wt.load package is a set of framework utilities for loading demonstration or test data. It can be expanded to load new classes. The following three classes can be used to run loads:

## **Demo**

Used to load demonstration data.

## **Developer**

Used to load initial users and access control rules. It is typically needed by a developer for using the Windchill system. The devuser.csv can be used to add initial users to the system.

## **LoadFromFile**

Callable from both the command line and from other classes. The Demo and Developer classes call this class to load much of the data that they load.

LoadFromFile is a framework that can be used to load data into your own site classes. The following source files are included as examples of how to use the load package: wt.load.Demo.java, wt.load.Developer.java, and wt.doc.LoadDoc.java.

LoadFromFile in the main or command line call, or the doFileLoad method call uses the following parameters to load data into the system:

### **data file**

Contains data to be loaded and resides on the MethodServer host. The data file is a required parameter. If the string is null or not on the command line, the load fails. It is recommended that you pass the full path name of the file. If the full path is not provided, however, the method attempts to read the file in the directory in which the load was started.

### **map file**

Used in the mapping of attribute names to sequence of data in the data file. It also contains the method to call to load the data. If the string passed is null, the method tries to use the value of wt.load.defaultMapFile from wt.properties. If that is also null, the default is wt.home\loadFiles\csvmapfile.txt where wt.home is from wt.properties. If there is no file to match, the load fails.

### **out file**

Not currently implemented.

### **token**

Used to delimit fields of data in the data file. If the token string is null, an attempt is made to get wt.load.defaultToken from wt.properties. If that also returns null, then ',' is used.

**method**

A string used with the object name (the first field) from the data file to key into the map file. The line in the map file then contains the method name to which this information is passed and the ordered list of field names to match with the line in the data file. If the input string for method is null, the default is 'create'.

**user**

Specifies the user to execute the method if no user is given in the line from the data file. The command line does not use the -u user option so no user is preset. The main in LoadFromFiles calls SessionHelper.manager.getPrincipal() to prompt for a user and perform the authentication. The doFileLoad still has the user passed in but it can be null. Demo and Developer authenticate before they call doFileLoad. It is recommended that you authenticate a user before calling doFileLoad, because it does not authenticate the user. The user would eventually be authenticated when the server side attempts to setPrincipal using the user passed in from doFileLoad, but the error messaging is best handled on the client side before calling doFileLoad.

To run LoadFromFiles from the command line, be sure that the ServerManager and MethodServer are running and then call:

```
java wt.load.LoadFromFile -d c:\ptc\Windchill\loadfiles\ReqDocs.csv
```

# Customizing Loading

You can customize loading by either changing the data in the data (or csv) files provided in the Windchill\loadFiles directory or adding new methods to load locally customized classes. Changing the data in the files loads local data into existing classes. The following sections describe the map file and the data file.

## Map File

Following is an example of a map file:

```
#class,method,real method~attribute 1~attribute 2~&hellip;.~attribute n
General~create~wt.doc.LoadDoc.createGeneral~user~name~title
  ~number~type~description~department~saveIn~path~format
~project~domain~lifecycletemplate1
```

The map file is used to map the specific method call and the data fields to a line in the data file. The first two fields, class and method, are the keys to the map file. Two keys are given to each line in the map file to allow multiple definitions for a given actual class in Windchill. This allows alternate names for one class, multiple functions for one class, or different input fields. Both class and method are arbitrary strings. The class value matches the class in the data file. It is used only to match the map and data files so it can be any string as long as the same string is used in both. The method value matches the method variable passed either on the command line or as a parameter to doFileLoad. The real method field is the fully qualified method name that load calls through introspection with the values from the data file. The attributes 1 through n match the text used in the real method. The order in the map file is used to retrieve the values from the data file.

The following are possible scenarios for modifying this file:

- Data for the data file has the data fields in a different order. Edit the csvmapfile.txt to make the order the same as the data file.
- You want to add a new site class. Create a new line in the map file with a new class, real method, and new attribute list. Following is an example line added to csvmapfile.txt for a new class:

```
NewClass~create~wt.doc.LoadDoc.createNewClass~user~name~title
  ~number~type~description~department~my_new_attribute~saveIn
~path~format~project~domain~lifecycletemplate
```

Do not put blank lines in the map file. One or more empty lines between entries cause the load to fail.

---

1. The preceding is only one line but wrapped here for readability. Continuation of lines is not accepted.

## Data File

Following is an example of a data file:

```
#Class,user,name,title,number,type,description,department,
  saveIn,path,format~project~domain~lifecycletemplate
General,phg,EGadWork,Econo Gadget R6.1 Work Plan,1,Document,
  Work plan for Econo Gadget R6.1,DESIGN,/Design/Gadget/Econo
  Gadget,EGadWork.xls,Microsoft Excel,,,
```

The # character as the first character of a line indicates a comment and the line is ignored. Blank lines and spaces at the beginning of a line are ignored. As in the earlier example, the preceding is only one line but wrapped here for readability. Continuation of lines is not accepted.

The data file is used to provide the load with data. The first field is the class field, which was just described under the map file description. The rest of the fields are strings to be mapped to the attributes listed in the map file. To load different data than was shipped with Windchill, create a new data file. If a new class is created as was just described under the map file description, a new data file would have to be created with records like the following:

```
NewClass,phg,EGadWork,Econo Gadget R6.1 Work Plan,1,Document,
  Work plan for Econo Gadget R6.1,DESIGN,my new attribute value,
  /Design/Gadget/Econo Gadget,EGadWork.xls,Microsoft Excel,,,
```

Note that, if you edit a data file, part numbers must be uppercase.

## Creating New Methods for Loading

All the methods that are called by the load follow the same signature:

```
boolean createNewClass(Hashtable nv, Hashtable cmd_line,  
    Vector return_objects )
```

The method should return true if the function is successful, and false if it is not. The parameters are as follows:

- The `nv` parameter is a hash table of the attribute names from the map file as the keys and the corresponding strings from one line of the data file as the value.
- The `cmd_line` parameter is a hash table of the extra attributes from the command line (parameters not defined in the interface, such as `&username` in the `devuser.csv` file). This allows substitution of values from the command line into the data file at runtime.
- The `return_objects` parameter is a vector of the objects created or worked upon by the `createNewClass` method. After the object is manipulated, add it to the vector; for example:

```
return_objects.addElement(doc);
```

This vector is used to give more informative messages back to the command line. If the object that was manipulated by this method does not implement `getIdentity` or that information would not be useful in a message to the command line, a string should be added to the `return_object` instead; for example:

```
String msg = "Type = Content Item, Identity =  
    " + appData.getFileName();  
return_objects.addElement(msg);
```

There is a utility that resolves the data from the data file and the data from the command line. For example, to retrieve a value for the new attribute that was added for `NewClass`, use the following:

```
String my_attribute = LoadServerHelper.getValue("my_new_attribute",  
    nv, cmd_line, LoadServerHelper.REQUIRED);
```

The first parameter is the string from the map file. The last parameter is used to indicate if the field is required, not required, or blank okay. If the field is required or the load should fail, use `LoadServerHelper.REQUIRED` and check if the return value is null. `LoadServerHelper.getValue` generates an error message if the value is required and the field value is null or an empty string.

If the field is not required and you want the return set to null for no value or empty strings, use `LoadServerHelper.NOT_REQUIRED`.

If you want to know the difference between no value being given for a field and an empty string, use `LoadServerHelper.BLANK_OKAY`. The blank okay option returns null if the attribute is not found in the hash table, meaning there could be a format issue with the map and data files. The blank okay option returns "" if the attribute is found in the hash table but the value was blank, which, depending on the attribute, could be okay.

There are three other useful utilities that can be used inside a method. The `LoadServerHelper.changePrincipal(user)` method can be used to change the effective user for a set of actions. This can be used only if the user who was authenticated for this session is a member of the administrative group. Upon completion of processing one line of data, the load utility sets the principal back to the original one, if the method it called changed the principal.

The `LoadServerHelper.removeCacheValue(MY_KEY)` method and `LoadServerHelper.setCacheValue(MY_KEY,my_object)` method can be used to cache objects between calls to a specific method or calls to different methods. The load utility calls one method per line in the data file. Loading documents and then loading multiple content files into a document is one example of first creating a document, caching it, and returning to the load. The load then reads in the next data line, which is a content line. The `LoadContent` method retrieves the cached data as follows:

```
ContentHolder contentHolder =  
    (ContentHolder)LoadServerHelper.getCacheValue(CURRENT_CH_KEY);
```

and adds the content file to the document. If you are creating an object that can hold content (files and URLs) and you want to load multiple content items on lines following the object in the data file, you must cache the object using the following constant:

```
private static String CURRENT_CH_KEY = "Current ContentHolder";
```

If you want to cache an object for another reason, you must create a unique key string. It is recommended that you clear your cached object at the beginning of the next create so that if the create fails, the next operation depending on it will fail and not corrupt other data. The cache of all objects is cleared after the last data line is processed.

The load utility creates a transaction block around the processing of each line of data. If the method returns false, the transaction block is rolled back. If it returns true, it is committed. The cache is not cleared if the method returns false.

The following is an example of what to add to `LoadDoc.java` to add the `NewClass` example. `LoadDoc.java` source is provided in `wt.doc`.

```
public static boolean createNewClass(Hashtable nv, Hashtable cmd_line,  
    Vector return_objects) {  
  
    boolean flag = true;  
  
    String user = LoadServerHelper.getValue("user",nv,cmd_line,  
        LoadServerHelper.NOT_REQUIRED);
```

```

String name = LoadServerHelper.getValue("name",nv,cmd_line,
    LoadServerHelper.REQUIRED);
if (name == null)
    flag = false;

String number = LoadServerHelper.getValue("number",nv,cmd_line,
    LoadServerHelper.REQUIRED);
if (number == null)
    flag = false;

String my_attribute = LoadServerHelper.getValue("my_new_attribute",
    nv,cmd_line,LoadServerHelper.REQUIRED);
if (type == null)
    flag = false;

if (flag == false)
    return false;

try {
    LoadServerHelper.removeCacheValue(CURRENT_CH_KEY);

    LoadServerHelper.changePrincipal(user);

    NewClass new_class = NewClass.newNewClass(number,name,
        DocumentType.toDocumentType("NewClass"));
    new_class.setMyNewAttribute(my_attribute);
    return persistDocument(nv,cmd_line,new_class,return_objects);
}

catch (PropertyVetoException pve) {
    LoadServerHelper.printMessage("Couldn't create NewClass Document:
        PVE exception caught.");
    pve.printStackTrace();
    return false;
}
catch (WTEException wte){
    LoadServerHelper.printMessage(wte.getLocalizedMessage ());
    wte.printStackTrace();
    return false;
}
catch ( Exception e ) {
    LoadServerHelper.printMessage("Error detected in
        createNewClass1");
    e.printStackTrace( );
    return false;
}
}

```

## Existing Load Methods

The best way to see what methods are available to load with on your system is to look at the `.loadFiles.csvmapfile.txt`. The following is a brief directory of the methods available for loading data in the system. Full method paths are given to allow you to look them up in javadoc.

- `wt.doc.LoadDoc.createGeneral`  
Loads General documents; adds one content file and caches the General created for additional content additions.
- `wt.doc.LoadDoc.createReq`  
Loads Requirements documents; adds one content file and caches the Requirements created for additional content additions.
- `wt.doc.LoadDoc.createSpec`  
Loads Specification documents; adds one content file and caches the Specification created for additional content additions.
- `wt.load.LoadContent.createContentURL`  
Adds a content URL to a cached content holder.
- `wt.load.LoadContent.createContentFile`  
Adds a content file to a cached content holder.
- `wt.load.LoadContent.setDefaultDirectory`  
Sets a default directory where content can be found. It is in effect until the next `setDefaultDirectory` or the end of the data file.
- `wt.load.LoadContent.setDefaultURL`  
Sets a default portion of the URL where content can be found. It is in effect until the next `setDefaultURL` or the end of the data file.
- `wt.folder.LoadFolder.createCabinet`  
Creates a cabinet. It is used mainly for initializing the system. User cabinets are created automatically when a user is created.
- `wt.folder.LoadFolder.createSubFolder`  
Creates a subfolder.
- `wt.folder.LoadFolder.createFolderShortcut`  
Creates a folder shortcut.
- `wt.load.LoadUser.createUser`  
Creates a new user.

- wt.load.LoadUser.createGroup  
Creates a new group.
- wt.load.LoadUser.createUserGroup  
Adds a user to a group.
- wt.load.LoadUser.createDomain  
Creates a new domain.
- wt.load.LoadUser.createAccessRule  
Adds an access rule.
- wt.part.LoadPart.createPart  
Creates a part.
- wt.part.LoadPart.addPartToAssembly  
Adds a part to an assembly in a "uses" relationship.
- wt.part.LoadPart.createPartDocReference  
Creates a "references" relationship between a part and an existing document.
- wt.index.LoadIndexRule.createIndexRule  
Creates index rules for searches.
- wt.lifecycle.LoadLifeCycle.createLifeCycleTemplate  
Creates a life cycle template.
- wt.project.LoadProject.createIndirectRoleHolder  
Creates a role-to-role mapping.
- wt.project.LoadProject.createRoleHolder  
Creates a role-to-principal mapping.
- wt.project.LoadProject.createActorRoleHolder  
Creates a role-to-actor role mapping. Actor roles are derived users, such as "Creator".
- wt.lifecycle.LoadLifeCycle.createCriterion  
Creates permission criterion for life cycle phase.
- wt.lifecycle.LoadLifeCycle.createPhaseTemplateBegin and  
wt.lifecycle.LoadLifeCycle.createPhaseTemplateEnd  
Creates a phase.

- `wt.project.LoadProject.createProjectBegin` and `wt.project.LoadProject.createProjectEnd`  
Creates a project.
- `wt.lifecycle.LoadLifeCycle.createWTAclEntry`  
Creates ad hoc access control for a life cycle phase.
- `wt.content.LoadDataFormat.createDataFormat`  
Adds data formats used in load content files.
- `wt.effectivity.ConfigurationItem.createConfigurationItem`  
Loads a configuration item. A configuration item is necessary for lot number effectivity and serial number effectivity, but is optional for date effectivity.
- `wt.effectivity.WTDatedEffectivity.createWTDatedEffectivity`  
Loads a date effectivity. The effectivity will be associated with the WTPart object that was loaded last.
- `wt.effectivity.WTSerialNumberedEffectivity.createWTSerialNumberedEffectivity`  
Loads a serial number effectivity. The effectivity will be associated with the WTPart object that was loaded last.
- `wt.effectivity.WTLotEffectivity.createWTLotEffectivity`  
Loads a lot effectivity. The effectivity will be associated with the WTPart object that was loaded last.



# 7

## Customizing the HTML Client

In Windchill PDM, the majority of HTML clients are produced using a PTC technology called template processing. A few clients are produced from JSP pages. PDM Link makes greater use of JSP pages and ProjectLink pages are almost all generated from JSP pages. Future releases of both PDMLink and ProjectLink will make use of PTC's new application framework called DCA. DCA is described in detail Windchill Client Technology Guide.

This chapter describes how to create and implement HTML clients using template processing. It covers the basic features used in both Windchill PDM and PDMLink. Extensions to template processing features specific to PDMLink are covered in the Customizing PDMLink Template Processing Clients chapter.

An overview of HTML client development is given first, followed by an example of creating a view page. The example gives step-by-step instructions but does not go into detail about general concepts. The sections that follow the example provide the details on how to implement template processors in general and how to use the HTML template factory. These sections are considered reference material for the first example.

Next is another step-by-step example of how to add an action to an HTML client. Again, this example is followed by reference sections with more detail about form processing and action processing.

You should go through both of these examples thoroughly before attempting to write or customize an HTML client. Code segments used in the examples can be found in the customization/HTMLClient directory, specifically in Example1.doc and Example2.doc.

The remaining sections provide both examples and reference information on using HTML help links, HTML components, tables, and the table service.

<b>Topic</b>	<b>Page</b>
Overview .....	7-3
Creating a View Page in the HTML Client with an Association—Example 1	7-14
How to Implement Template Processors .....	7-23
How to Use the HTML Template Factory .....	7-41
Adding an Action in the HTML Client—Example 2.....	7-47
Form Processing and Action Processing.....	7-61
Properties and Property Files .....	7-71
Adding an HTML Help Link .....	7-85
Overview of HTML Components, Table, and Table Service.....	7-92
Using HTML Components .....	7-102
Using HTML Tables and Custom Tables .....	7-111
Using the HTML Table Service.....	7-119
Performing a Multiselect in the HTML Client .....	7-126
Changing the Presentation of the Attributes in the HTML Client.....	7-133
Setting Color and Style Attributes in HTML and JSP Clients.....	7-142
HTML Component Implementation .....	7-146
HTML Table Generation and Table Service Implementation .....	7-154

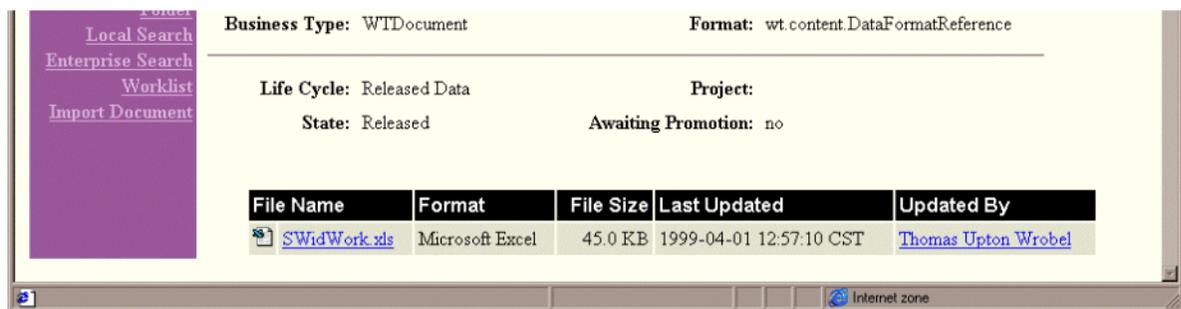
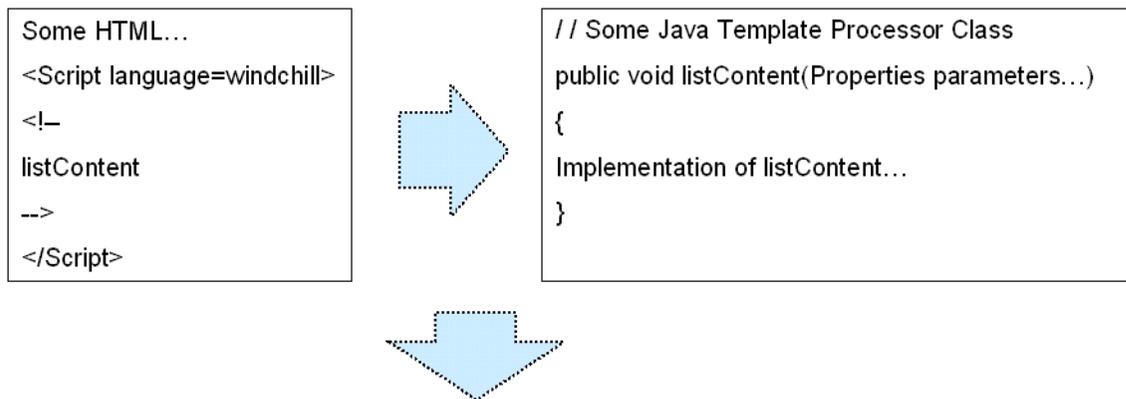
# Overview

This section gives an overview of how HTML clients are developed. Specifically, a view page (that is, a view-only page that displays information and is non-interactive) and a page containing a form (with which the user interacts and causes some processing to be done) will be discussed.

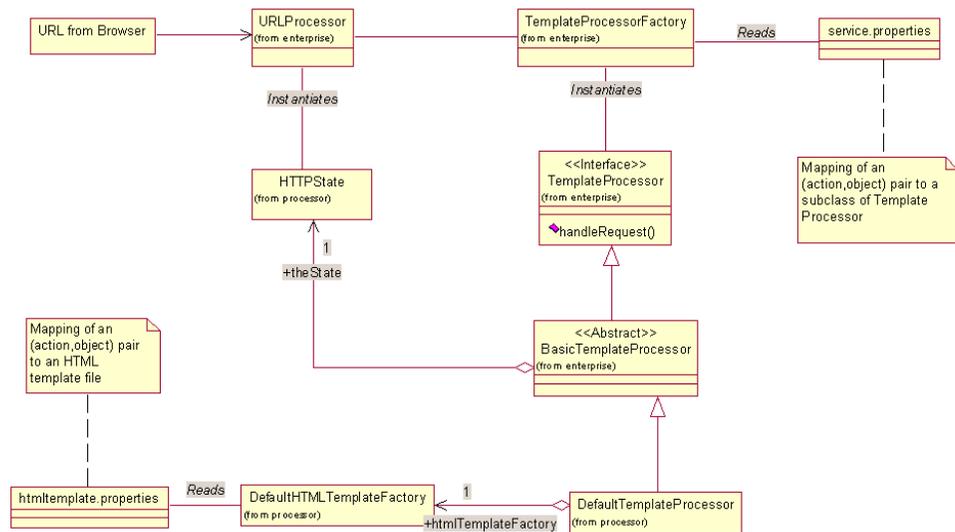
HTML template processing refers to the architecture by which information is accessed on the server and is displayed in HTML on the client. There are essentially two parts to this architecture:

- An HTML template file, which contains HTML text, may contain Javascript, and may contain what is referred to as Windchill script
- A Java class called a template processor

As shown in the following figure, a Windchill script call exists in an HTML template file. A Java template processor processes the HTML template file and the Windchill script call in the HTML template causes the corresponding method in the Java template processor to be invoked. The script call in the HTML template is replaced by the output of the method which is then displayed on an HTML page.



The following figure shows a combination of the process and the structure.



In a Web browser, a user clicks on a link and the URL is sent to the URLProcessor class. URLProcessor is the gateway into the template processing architecture. In an HTML client, almost all URLs go through URLProcessor.

URLProcessor calls the TemplateProcessorFactory class. TemplateProcessorFactory maps information from the URL to entries in the services.properties file to find the correct instance of the TemplateProcessor interface to return. Basically, the service.properties file contains a mapping of an action and possibly an object to a specific TemplateProcessor. TemplateProcessorFactory then returns the correct TemplateProcessor to URLProcessor.

URLProcessor also creates and initializes an HTTPState object. This object contains information from the URL which both the TemplateProcess and URLProcessor can use. This mechanism essentially allows state information to be passed.

BasicTemplateProcessor is a class that contains several service methods you can use, for example, to get the icon for a business object or a class, and to display the contents associated with a content holding object.

DefaultTemplateProcessor is the TemplateProcessor subclass we recommend you extend when you create new template processors. It has the HTTPState cookie and provides a default implementation of the steps performed by a TemplateProcessor, thus hiding many of the details of the actual processing of an HTML template file and the locating of the HTML template file to be used.

DefaultTemplateProcessor works with DefaultHTMLTemplateFactory class. DefaultHTMLTemplateFactory is similar to TemplateProcessorFactory in that it looks up a mapping in the htmltemplate.properties file to find the HTML template file to be processed.

The detailed process for displaying view pages using template processing is as follows:

1. A URL arrives at the Windchill server, for example:

```
http://clancey.mn.ptc.com/wtcgi-bin/wtauthcgi.exe/  
wt.enterprise.URLProcessor/URLTemplateAction?action  
=ObjProps&oid=vr%3awt.doc.WTDocument%3a111235
```

The URL contains three pieces of information that are used:

- The server address, which can be CGI-based or a servlet, for example:

```
http://clancey.mn.ptc.com/wtcgi-bin/wtauthcgi.exe
```

- The class and the specific method in the class that is going to be invoked to process the URL, for example:

```
wt.enterprise.URLProcessor/URLTemplateAction?
```

The `wt.enterprise.URLProcessor` class is a gateway class. `URLTemplateAction` is a static method in that class which will be invoked.

- The query string, for example:

```
action=ObjProps&oid=vr%3awt.doc.WTDocument%3a111235
```

The query string contains name/value pairs that are passed to `URLProcessor` and also to `TemplateProcessor`. If you need to communicate information to either processor, you can pass it on the query string.

2. The `URLProcessor` performs the following actions:
  - a. Reads the query string and Post data.
  - b. Processes the action and the OID from the query string. If you pass in an OID on your URL, `URLProcessor` converts the OID back into an actual business object.
  - c. Creates and initializes the `HTTPState` object using information passed on the query string. This object acts as a cookie storing the state of the process. This state information can be used by the `TemplateProcessor`.
  - d. Uses `TemplateProcessorFactory` to find the particular instance of a `TemplateProcessor` that will handle the processing. `TemplateProcessorFactory` maps the action and OID/class to an entry in `service.properties` and returns the corresponding `TemplateProcessor` subclass to `URLProcessor`.
  - e. Passes the `HTTPState` object and the process to the instance of the `TemplateProcessor` that was returned from `TemplateProcessorFactory`.

3. The TemplateProcessor that was found by the TemplateProcessorFactory performs the following actions:
  - a. Processes the query string and form data to retrieve information to set local variables to be used in the template processing. The TemplateProcessor is defined to expect the information it is passed. For example, if you pass three context objects, the TemplateProcessor expects all three and gets that information by accessing the HTTPState object.
  - b. Uses DefaultHTMLTemplateFactory to find the desired HTML template file. Using the same context information, DefaultHTMLTemplateFactory maps the action and OID/class (if they exist) to entries in htmltemplate.properties to find the proper HTML template file.
  - c. Parses the HTML template file to generate the HTML output to the browser using the following call:

**template.process** (*os*, *this*)

In this call, *os* is a Java OutputStream and *this* is a reference to the TemplateProcessor making the call.

When the HTML template file is parsed, the TemplateProcessor reads through the file, ignoring any HTML text and Javascript text, and instead passing that text through to the output. If it encounters a Windchill script call, however, the corresponding Java method in the TemplateProcessor is invoked. The script call in the HTML template is then replaced with the output of the method invoked in the TemplateProcessor. For example, the following code in the HTML template file:

```
<B>
<Script language=Windchill>
<!--
objectPropertyValue propertyName=Creator
-->
</SCRIPT>
</>
```

would be replaced with the following code:

```
<B>
Administrator
</B>
```

Windchill script calls in HTML template files can be of two types:

- Static, in which case you specify the fully qualified class name and the method to be invoked (which must be a static method). For example:

```
<Script language=Windchill>
<!--
wt.enterprise.MyTemplateProcessor addMyInfo verbose=true
-->
</SCRIPT>
```

The corresponding method in the Java `TemplateProcessor` must be defined as a static method with a void return type and it must have the three parameters shown in the following syntax:

```
public static void addMyInfo(Properties parameters, Locale locale,
OutputStream os)
```

- Instance-based, in which case you specify the method to be invoked. For example:

```
<Script language=Windchill>
<!--
addMyInfo verbose=true
-->
</SCRIPT>
```

The corresponding method in the Java `TemplateProcessor` is not a static method, but it returns void and it must have the three parameters shown in the following syntax:

```
public void addMyInfo(Properties props, Locale local, OutputStream os)
```

The parameters passed have the following meanings:

#### **Properties**

Properties object; it contains name/value pairs that appear after the method call in the HTML template file. They are essentially parameters to the script call. No space is allowed between a key and a value, but spaces are allowed between additional parameters.

They are passed to the Java method as strings, so you must define the Java method to expect them as such. For example, if you want to handle a flag in your method, the method must look for a key named "Verbose".

#### **Locale**

Locale of the client browser that is displaying the HTML page. This parameter is used if you need to look up information in resource bundles or handle any localization of output text.

#### **OutputStream**

The actual output stream passed by the server to be used as the output stream for text; this is where the HTML is written.

If you do not have your method signature defined to return void and have these three parameters, the method to be invoked will not be found when that HTML template file is processed and that script call is encountered.

There are no formal rules on how you define your method signature to handle exceptions. You can define it to throw any exceptions it encounters or handle all the exceptions it encounters. Service methods exist in `BasicTemplateProcessor` for displaying the exceptions that are thrown when you are processing a template file. Most script calls throw these exceptions and you can display them on a resulting page.

The following are the general steps to build a view page and include it in the HTML client:

1. Subclass `DefaultTemplateProcessor`.
2. Add methods to your subclass of the `DefaultTemplateProcessor` subclass which correspond to script calls you may want in your HTML template file.
3. Create the HTML template file to be processed.
4. Add a property so the appropriate action/object pair can be mapped to your subclass of `DefaultTemplateProcessor`. Note the values of the action/object (oid) pair in the following sample URL:

```
http://MyServer/wtcgibin/wtauthcgi.exe/  
wt.enterprise.URLProcessor/  
URLTemplateAction?action=ReferencedBy& oid=  
VR:wt.doc.WTDocument:1234
```

For this URL, you must have a property that looks like the following:

```
wt.services/svc/default/wt.enterprise.TemplateProcessor/  
ReferencedBy/wt.doc.WTDocument/  
0=wt.mystuff.MyTemplateProcessor/  
duplicate
```

Note the specification of the action (`ReferencedBy`) and the oid (`wt.doc.WTDocument`) in this entry.

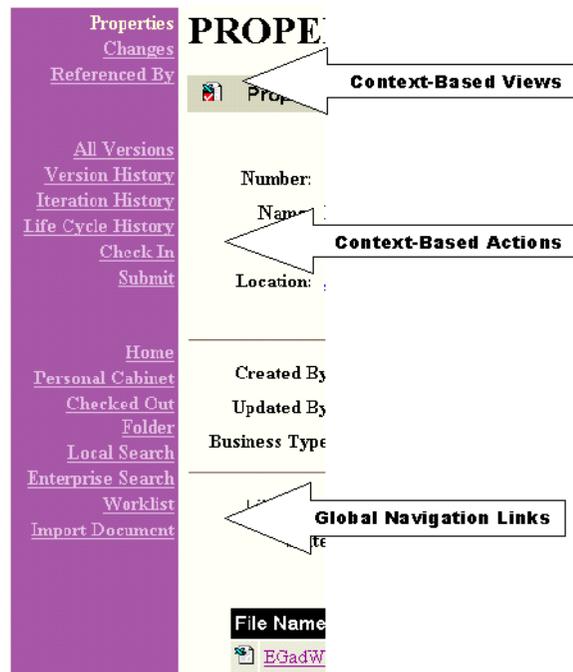
**Note:** See the section, `Properties and Property Files` for information on how to add new properties to the system.

5. Add a property so the action/object pair can be mapped to your HTML template file. Again, assuming the preceding URL, you must also have a property that looks like the following:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/  
ReferencedBy/wt.doc.WTDocument/0=templates.doc.referencedBy
```

**Note:** See the section, `Properties and Property Files` for information on how to add new properties to the system.

- Put a link in your HTML client so the new page can be navigated to. There are three sections in the navigation bar in the HTML page; each section represents a type of link.



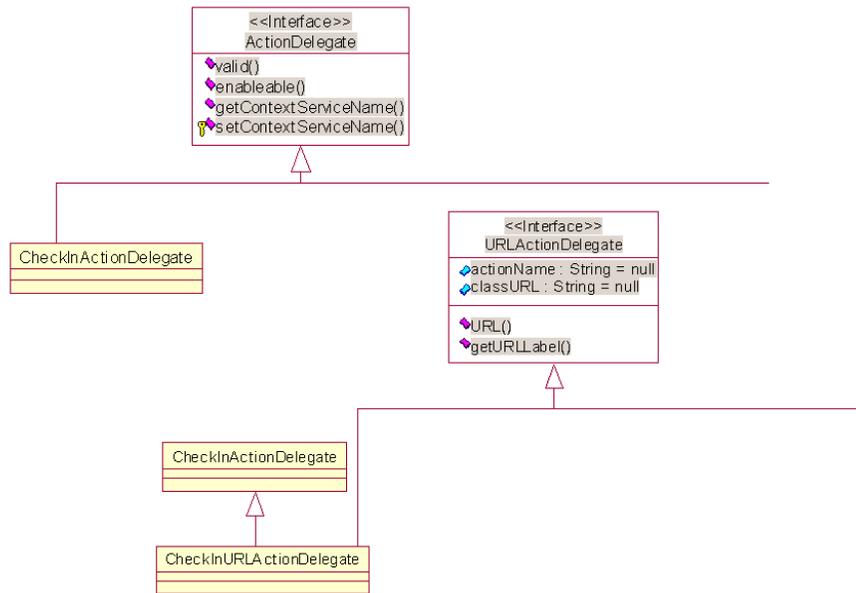
- To add a context-based view link, edit `wt.enterprise.URLLinkResource` to add the new link.
- To add a context-based action link, edit `wt.enterprise.URLLinkResource` to add the new link. Then create a new `ActionDelegate/URLActionDelegate` (discussed in more detail next in this section) pair, if needed. Finally, put an entry in `service.properties` for the `ActionDelegateFactory` and the `URLActionDelegateFactory` to find your new delegates.
- To add a global navigation link that is available independent of what is presented on the page, modify the method `createGlobalNavigateLinks` in `wt.enterprise.BasicTemplateProcessor`.

The `ActionDelegate` class checks both the permission of the user to perform the action and the availability of the action. For example, is the user allowed to check out a part and is the part available for checking out?

Action delegates are completely independent of any user interface. They are implemented to control whether a menu option is available in the navigation bar when a particular object is displayed. They have no knowledge of how something is going to be displayed.

URLActionDelegate is a subclass of ActionDelegate and is used specifically for generating links in HTML pages. It has the logic to check whether an action should be available and has methods to actually generate a URL to a particular action. It has methods to get a label to be displayed as a link for invoking that action and methods to get a string representation of the URL to that particular action.

Because both classes are modeled, you can create a new ActionDelegate or URLActionDelegate simply by opening Rational Rose and extending from the parent classes. For URLActionDelegate, also extend from an ActionDelegate. (See the following figure.) Generate code and then fill in the desired methods.



The first part of this section has discussed how to add and process pages that simply present views of information. The remainder of this section will describe how to support user interaction with forms, how to manipulate forms, and how to manipulate business objects, for example, creating business objects and checking them in and out. Generally, these kinds of actions are dependent on access control and the state of an object.

The `URLProcessor` class has three methods that are typically used for generating forms, processing forms, and invoking actions:

#### **generateForm**

Creates HTML pages that have HTML forms. This method is invoked to display a form. It involves using an `ActionDelegate` to check whether the user should have access to the action and looking up a `TemplateProcessor` to display the form.

#### **processForm**

Handles processing of the HTML forms that are returned. This method looks up a form task delegate, which does the actual processing of the form.

Typically, `generateForm` and `processForm` are used together; you have a URL which invokes `generateForm` and the result of `generateForm` is a form whose action contains a call to `processForm`.

### **invokeAction**

Handles URLs that represent an action to perform (for example, checking out a document or deleting an entry) without user interaction. This method is similar to `processForm`. It gets an `ActionDelegate` and checks the access for a particular form. It also gets a form task delegate to actually do the processing of the form.

The difference between `processForm` and `invokeAction` is that `invokeAction` is typically used when no form information exists to be submitted; that is, when there is no user interaction involved with performing the action. For example, if you want to check out an object, typically you do not need any user input; you just need to follow the link to initiate the checkout. Use `invokeAction` if you do not need any input from the user.

Following are the general steps to create an HTML form:

1. Subclass `GenerateFormProcessor`.
2. Create an HTML template file that contains the form definition, that is, the fields of the form that you want to display.
3. Generate a link in the HTML client that will call the `URLProcessor` and invoke the `generateForm` method on `URLProcessor`. Pass in the context information (the action and the class) that you are dealing with, similar to the following:

```
...URLProcessor/generateAction?action=create&class=
wt.doc.WTDocument
```

4. Add a property to map the action/class to your `TemplateProcessor`.
5. Add a property to map the action/class to your HTML template.

Several general purpose methods for building a form are implemented in `GenerateFormProcessor`. By subclassing this class, you get access to the following convenience methods:

### **enumeratedTypeMenu propertyName=*attribute name***

Builds a drop-down list based on the options of the enumerated type that is specified by the attribute name. (The *attribute name* specified must be an attribute that is an enumerated type.) For example, in the Create Document form, a drop-down list allows the user to pick a department, which is an enumerated type.

### **contextualValue**

Places a `textField` form element, which can be used to output an attribute of a value.

### **listLifeCycles**

Builds a drop-down list showing the available life cycles.

A JavaScript method is also available that you can use to add support for browsing the cabinet/folder hierarchy and selecting a folder.

After you have generated your form, you use `generateForm` to actually display the form. Your form contains an action which is a URL to invoke `processForm` on the `URLProcessor` class. The URL would look similar to the following:

```
...URLProcessor/processForm?action=...&class=...&
```

In this URL, the action argument is used to find the `ActionDelegate` and `FormActionDelegate` (described next in this section). Either the oid or class argument is used to locate the `FormActionDelegate` (also described next in this section).

The preceding URL is generated in the HTML template as follows through a call to `generateAction`:

```
<form name="importDocument" enctype="multipart/form-data" action=
<SCRIPT LANGUAGE=Windchill>
getURLProcessorLink method=processForm action=navigateFolders
</SCRIPT>
method=POST>
```

The `getURLProcessorLink` method in `BasicTemplateProcessor` allows you to output the string by specifying it as a script call in the template file that builds your form. Specify the method you want to invoke and the action. Any other name/value pairs you include in the script call are added to the query string. Thus, you can use this script call to actually build the form action.

If you are sending files, the HTML form element must specify the `enctype=multipart/post` attribute. If you are sending content (uploading files), place the HTML FORM file inputs at the end of the HTML FORM. You must also send the content in contiguous blocks.

The actual processing of the form is done with `FormTaskDelegateFactory` and `FormTaskDelegate`. `FormTaskDelegateFactory` is similar to the other factories used in HTML template processing. It takes context information from `URLProcessor` and maps it to an entry in `service.properties` to find the `FormTaskDelegate` instance to be returned. You can pass in an action and an object, or an action and a class, as context information.

The `FormTaskDelegate` actually does the processing of the form when it is submitted. When a form is submitted, `URLProcessor` invokes the method `processAction` in the `FormTaskDelegate`. This method has the following form:

```
public void processAction(ContentHTTPStream contentStream)
```

The `processAction` method has one parameter, `contentHTTPStream`. This parameter reads in and processes files sent from the browser so it is used only if you are uploading files in your form. If not, the parameter is typically ignored.

Like the template processors, `FormTaskDelegate` also has access to the `HTTPState` object, which is how a form gets access to all of the form data. When `URLProcessor` creates the `HTTPState` object, it initializes the object with all of the

form data that is submitted in the form. The `FormTaskDelegate` uses the `HTTPState` object to get access to any form data.

The `processAction` method is responsible for several actions:

- Validating the query string/form data that was sent on the form. For example, if you are creating an object and a name is required, it might validate that a name was given.
- Performing the desired action. For example, if you are creating an object, it actually does the creation, saves it in the database, and so on.
- Updating the context information, setting the action/object or action/class based on the results of the data validation and performing the action. This decides what response page is generated.

For example, if you make a form to create an object, it will validate the input, create the object, and set that newly created object as the context object. This is important because the context information controls what is displayed after your form is submitted.

The general flow of events in form/action processing is as follows:

1. A URL arrives at the server.
2. The URL goes through `URLProcessor`, which uses `FormTaskDelegateFactory` to find the correct instance of `FormTaskDelegate`. `URLProcessor` also initializes the `HTTPState` object and passes it off to the `FormTaskDelegate`.
3. `FormTaskDelegate` validates the data, performs the action, and then sets the context based on the validated data and action.
4. Control returns to `URLProcessor`, which uses the context information to find the `TemplateProcessor` that will display the resulting page.
5. The `TemplateProcessor` then generates a response page to send to the browser.

In the case of `Create Document`, once the form is submitted, a `FormTaskDelegate` creates the document and sets the context information so that when it returns to `URLProcessor`, the result is a properties page for the object being displayed.

## Creating a View Page in the HTML Client with an Association—Example 1

This example shows how to create and add a new View page to the HTML client. The objectives of this example are as follows:

- To describe the basic principles of adding an HTML page to the HTML client, the most important Java classes that you must create and modify, and the property files you must modify.
- To describe how to add the view of an association to an HTML page.

The three major steps of this example are as follows:

1. Create an HTML template file with calls to a template processor and calls to a template processor service.
2. Create a subclass of `wt.templateutil.processor.DefaultTemplateProcessor` that supports the calls within the new HTML template.
3. Add a link to the navigation bar in the HTML client.

### Before Starting

Before starting this example, open your `wt.properties` file and add the following line as the last entry in the file

```
wt.template.cache.enabled=false
```

Save and close the file. It is helpful to use this setting temporarily for development, but it should be left as "True" in a production system.

Locate the file `customization/HTMLClient/Example1.doc`; it contains codes that you will want to copy to complete this example.

### Creating the HTML Template File

To create the HTML template file, perform the following steps:

1. From the `customization/HTMLClient/Example1.doc` file, copy the text under HTML Template Skeleton (also shown below) and paste it into a new HTML file.

```
<HTML>

<SCRIPT LANGUAGE=Windchill>
<!--
  wt.htmlutil.HtmlUtil createBase
-->
</SCRIPT>

<HEAD>
```

```

<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
  charset=<SCRIPT LANGUAGE=Windchill>getCharsetEncoding</SCRIPT>">
<TITLE>
  Properties of
  <SCRIPT LANGUAGE=Windchill>
  <!--
    objectPropertyValue propertyName=displayIdentity
  -->
  </SCRIPT>
</TITLE>

<STYLE TYPE="text/css">
  <!--
    FONT,H1,H2 {font-family: <SCRIPT LANGUAGE=Windchill>getWCFontFamily quotes=
      none</SCRIPT>}
  -->
</STYLE>

</HEAD>

<BODY BGCOLOR=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=bg-body</SCRIPT>
  TEXT=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=f-body</SCRIPT> >

<TABLE BORDER=0 WIDTH=100%>
  <TR ALIGN=LEFT>
    <TD ALIGN=RIGHT VALIGN=TOP WIDTH=15%
      BGCOLOR=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=bg-navbar</SCRIPT>
      CELLPADDING=0 CELLSPACING=0 >

      <TABLE WIDTH=100% CELLPADDING=0 CELLSPACING=0>
        <TR ALIGN=RIGHT>
          <TD>
            <SCRIPT LANGUAGE=Windchill>
            <!--
              createNavigationBar currentPage=<Action>
            -->
            </SCRIPT>
          </TD>
        </TR>
      </TABLE>

    <P>
    <BR>

    <TABLE WIDTH=100% CELLPADDING=0 CELLSPACING=0>
      <TR ALIGN=RIGHT>
        <TD>
          <SCRIPT LANGUAGE=Windchill>
          <!--
            createActionsBar
          -->
          </SCRIPT>
        </TD>
      </TR>
    </TABLE>

    <P>

```



```
<!-- Add Association Script below here -->

</TD>
</TR>
</TABLE>

</BODY>

</HTML>
```

2. In the new template file that contains the preceding code, search for the Windchill script method `createNavigationBar` and replace `<Action>` with a name of your choice. The name of the action is completely up to you, but it should have some relation to the page that the link leads to.
3. Search for the HTML comment `<!-- Add newPageMethod below here -->` and add a Windchill script call, `newPageMethod`. Use the same Windchill script syntax that is present in the page.
4. Save the template file in any directory under codebase with any name you like. The only requirement is that the name must have the standard HTML extension, either `.htm` or `.html`.

## Registering the HTML Template

To register the HTML template file you just created, you will need to create the following new property. See the section, Properties and Property Files for information on how to add new properties to the system.

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
<Action>/wt.part.WTPart/0=<FILE_NAME>
```

- a. Replace `<Action>` with the name of the action you selected in step 2 of creating the HTML template file.
- b. Replace `<FILE_NAME>` with the relative path from codebase where you saved your new HTML template in step 4 of creating the HTML template file. You must use the period (`.`) as a directory separator and leave off the file extension. For example, the file name:

```
templates/ObjectProperties/WTPart.html
```

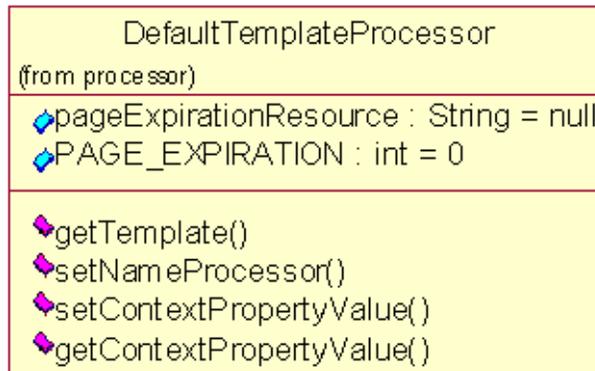
should be entered as:

```
templates.ObjectProperties.WTPart
```

## Creating the Template Processor

To create the template processor, perform the following steps:

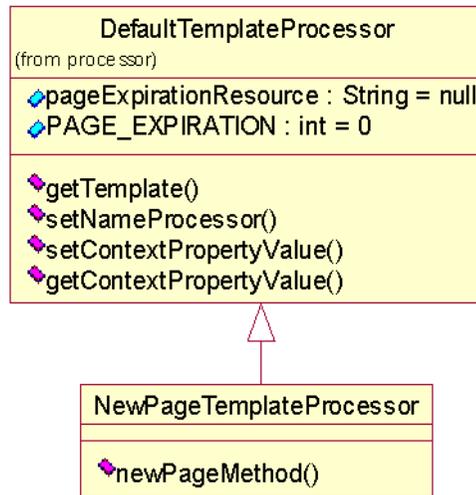
1. Open Rational Rose and load /src/WTdesigner.mdl. You will be modeling your template processor as a subclass of `wt.templateutil.processor.DefaultTemplateProcessor`.
2. Open the package `training.clientlab` under Logical View. If this package does not exist, you must add it.
3. Add a new Class Diagram under `training.clientlab`. Name the class diagram `Add Page`.
4. Add a new class to the `Add Page` diagram. Name the class `DefaultTemplateProcessor`. When you are done, your class diagram will have the following class:



5. Add a new class named `NewPageTemplateProcessor`. Then have your new class, `NewPageTemplateProcessor`, subclass `DefaultTemplateProcessor`.
6. Add a new method to the `NewPageTemplateProcessor` class. This method must generate the following signature:

```
public void newPageMethod(Properties parameters, Locale locale,  
    OutputStream os)
```

Be sure to include all three parameters in the method signature and use the correct modeling syntax. After you add this method, the diagram should look as follows:



**Note:** The newPageMethod does not show the parameters unless you click on it.

7. Generate the training.clientlab package by giving the NewPageTemplateProcessor class the focus. You should now have a template processor, NewPageTemplateProcessor.java, in the package training.clientlab.
8. The method stub for newPageMethod that was generated by the Rose model should have the following signature:

```
public void newPageMethod(Properties parameters, Locale locale,
    OutputStream os)
```

Use the following code (also found in customization/HTMLClient/Example1.doc under the label newPageMethod Body) to fill in the body of the method.

```
java.io.PrintWriter out = getPrintWriter(os, locale);
out.println("<H3>You have Part :");
out.println( ( (wt.part.WTPart)getContextObj() ).getName() );
out.println("</H3>");
out.flush();
```

9. Save the file.
10. Compile the new Java file. You now have a new template processor.

## Registering the Template Processor

To register the template processor, you will need to create the following new property. See the section, Properties and Property Files for information on how to add new properties to the system.

```
wt.services/svc/default/wt.enterprise.TemplateProcessor/  
<Action>/wt.part.WTPart/0=  
clientlab.NewPageTemplateProcessor/duplicate
```

Replace <Action> with the name of the action you selected in step 2 of creating the HTML template.

## Adding a Link for Your New Page to the HTML Client

To add a link for the new page to the HTML client, perform the following steps:

1. Open wt/enterprise/UrlLinkResource.java in your favorite editor.
2. Extend the WTPART\_ACTIONS entry in the resource bundle by adding the following text:

```
<Action>:<Action Display Name>
```

<Action> is the name of the action you selected in step 2 of Creating the HTML Template File. Replace <Action Display Name> with the text you want to appear in the browser as the hyperlinked text.

3. Save the file.
4. Compile. UrlLinkResource now has a new link for Documents in its listing.

At this point, the new page should be available for display.

## Finding Your New HTML Page

To verify that your new HTML page exists, perform the following steps:

1. Stop and restart the method server and the server manager.
2. From the Windchill home page, go to the Local Search.
3. Perform a search on Parts.
4. When the results come back, click the number of any of the parts. This should take you to the object properties page of the selected part.
5. A link with the text that you entered in the UrlLinkResource to replace the <Action: Display Name> should now appear in the first section of the Navigation bar. Click that link to return your new page.

## Adding a View of Associations to Your Page

To add a view of associations to the new page, perform the following steps:

1. Open the HTML template file that you created earlier in this example.
2. Search for the HTML comment `<!-- Add Association Script below here -->`. At this point, add the following Windchill script section (found also in `customization/HTMLClient/Example1.doc` under the heading `Windchill Script call for Associations:Part I`):

```
<SCRIPT LANGUAGE=Windchillgt;
<!--
tableService action=initAssociationNavigation ROLE=uses
  LINKCLASSNAME=wt.part.WTPartUsageLink
  OTHERSIDECLASS=wt.part.WTPartMaster
tableService action=initAssociationTable othersideattributes=name,
  number,type linkattributes=modifyTimestamp,quantity
tableService action=show
-->
</SCRIPT>
```

3. Save the HTML template.
4. Refresh your new page in the browser.

## Customizing the View of Associations

To customize the view of associations, perform the following steps:

1. Open the HTML template file you created earlier in this example.
2. Replace the Windchill script calls for adding the view of the associations with the following new set of calls (also found in `customization/HTMLClient/Example1.doc` under the heading `Windchill Script call for Associations:Part II`). Note the additional lines which allow in-place specification of the formatting of the HTML template.
3. Save the HTML template.
4. Refresh your new page in the browser. You should now see the changes in the presentation of the HTML table.
5. Enter the following command (also found in `customization/HTMLClient/Example1.doc` under the heading `Class Explorer Invocation`):

```
java wt.clients.beans.ClassExplorer wt.part.WTPartMaster
  wt.part.WTPartUsageLink
```

The Class Explorer Frame should pop up with two classes in the left panel.

6. You can now search under each class presented in the Class Explorer for available attributes to present in the table of the associations.
7. Select some new attributes for each class and add them to the list of attributes to display by adding them to the HTML template. You can also remove some of the attributes that are currently displayed.

8. Save the HTML template.
9. Refresh your new page in the browser. You should now see the new attributes in the HTML table that is displayed.

## How to Implement Template Processors

The preceding example showed a step-by-step process for creating a new template processor, but for a very specific purpose. This section gives more detailed, but more generic, descriptions of how to write and implement a template processor intended to present a view of information. A processor such as this would be used, for example, to present the properties of a business object or to present the iteration history of a business object. (This section does not describe how to generate HTML FORMs or how to process HTML FORMs when they are returned through an HTTP POST; these topics are described later in the chapter.)

The purpose of this package is to provide a tool for developers of HTML-based Windchill clients to simplify creating a template processor and processing an HTML template. This is done by allowing the use of the Rational Rose modeling tool to extend a default implementation of a template processor. Aspects of the template processor that vary from processor to processor are separated out by making method calls, either to an internal method or to an interface. Your subclass of the `DefaultTemplateProcessor` can either override the default implementation of a method call or an interface that is a plug-in point for customizing the template processor's behavior.

The most common activities to perform in the HTML client are to add a new page and to add a new subclass of `wt.enterprise.TemplateProcessor` (`wt.enterprise.BasicTemplateProcessor`, in practice) to support processing the new HTML templates that generate the page. The points that vary among the subclasses of `wt.enterprise.TemplateProcessor` are very specific. This package provides for a process and code that allows that developer to implement their points of customization while being able to rely on a default implementation for the other aspects of the implementation.

The remainder of this section provides first some background on the `HTTPState` class and general comments on modeling files for template processing. These are followed by an example of the default template processor implementation and several examples showing how to further customize it. Following the examples is more detailed, supplementary information on implementation.

### The `HTTPState` Class and Template Processing

#### Overview of `HTTPState`

The `HTTPState` class is used as a cookie to store the context of the template processing and provide some convenience services. The context of template processing is defined with the following elements:

- The current action
- The current object (if any)
- The class (if any)

These three pieces of information are used by the various factories that are used during template processing. These three fields control the direction of the template processing. The HTTPState class has fields for each of these three pieces of information, and has getters and setters for them.

Some of the convenience services that are provided by the HTTPState object are as follows:

- The original QueryString name/value pairs are available using the `getQueryData()` method. These values are set in the URLProcessor class.
- The original FORM data name/value pairs are available using the `getFormData()` method. These values are set in the URLProcessor class also.
- A running list of messages for either a header or a footer in the response page is kept. Corresponding service methods in BasicTemplateProcessor print out these accumulated lists of headers or footers.

The Windchill script APIs `showResponseHeaders()` and `showResponseFooters()` enumerate over the WTMessages that have been added to the running total using the HTTPState calls `addToResponseHeaders(WTMessage newHeader)` and `addToResponseFooters(WTMessage newHeader)` respectively. After printing out the headers or footers, the running total is cleared. The method in BasicTemplateProcessor that formats the output of the localized messages is as follows:

```
public void showResponseMessage( String message,  
Properties parameters, Locale locale, OutputStream os)
```

By overriding this method in your own template processor, you can specify the format of the presentation of the localized messages.

- A running list of exceptions is also kept. This useful if there are several non-terminal exceptions that you would like to pass back to the user. Just like the two preceding cases, there is a Windchill Script API and a corresponding call in HTTPState to add exceptions: `showResponseExceptions()` and `addToResponseExceptions( WTMessage newHeader )`. The method calls have the same effect for exceptions as for headers and footers.

Any TemplateProcessor that subclasses `wt.enterprise.BasicTemplateProcessor` has an aggregated instance of an HTTPState object. The `getContextObj()` and `setContextObj()` methods are actually proxy calls to the HTTPState object and its fields. The purpose of this is that if the developer wants to call a service that requires the state/context of the process (such as processing a subtemplate), then passing off the HTTPState object is all that must be done. Also, the HTTPState object is used as the communication conduit between the FormTaskDelegate that performs an action and the TemplateProcessor that generated the ensuing response page.

## HTTPState and URLProcessor

The URLProcessor does some initial handling of the QueryString and the Form data so that it can use several factories required for template processing. During the initial handling of the data, the URLProcessor instantiates an HTTPState object and initializes the values of the contextAction, the contextObj, and the contextClassName based on the values passed in on the query string and the Form Data.

After an instance of a TemplateProcessor is returned from the TemplateProcessorFactory, that instance of the HTTPState object is passed in on the bizData field of the HTTPRequest object. In the case of the DefaultTemplateProcessor, this passed-in HTTPState instance is retrieved and used to set the state of the DefaultTemplateProcessor. Existing processors can easily be rewritten to retrieve and use the HTTPState object with the following line in the handleRequest method:

```
setState( (HTTPState)request.bizData );
```

The benefit of this is that the TemplateProcessor is not duplicating what URLProcessor already did. In particular, if a persisted object was retrieved from the database based on an oid that was passed in the queryString or Formdata by URLProcessor, then that object is available from the passed-in HTTPState object. The database need not be accessed a second time.

When a URL arrives to the server and is handed off to one of the methods in URLProcessor to start a template processing activity, an HTTPState object is instantiated. URLProcessor uses several factories:

- TemplateProcessorFactory
- FormTaskDelegateFactory
- ActionDelegateFactory

These factories select the correct class to perform an action. Each of these factories uses either an action/object pair or an action/class name pair to select the desired class. The URLProcessor gateway methods parse the query string and the form data to retrieve this information so that it can use the factory to find the desired class. The information generated during this process is used to initialize the HTTPState object.

## General Comments on Modeling Files for Template Processing

When using Rational Rose to model the classes that you will use in HTML template processing activities, you must perform the following actions:

- For the classes that you model, go to the Windchill tab and set Serializable to None.
- For the attributes that you model, set the following in the Windchill tab for the attribute (not the class):
  - Set Persistable to False.
  - Set Bind to False.
  - Set Constrain to False.

If you do have constraints on your variables, you might want to set these to true. But most of the time you will want them set to false.

## The Default Implementation

This example illustrates the default implementation of the template processor. The purpose is to create an HTML page that is a view of information for a specific business object. The view of information is not dependent on the state of the object or the permissions of the viewer.

The following are pre-conditions for this example:

- You need only present a valid URL/PostData to set the context and the information for generating the view.

All template processing is required to have an action passed in either the QueryString or the HTTP POST data. The OID is used by the wt.fc.ReferenceFactory to retrieve the specific business object from persistent storage. You do not need any other information sent to generate the presentation of your page.

- There are no contingencies on the state of the business object (for example, it need not be checked in or submitted) to present the view.
- There are no access restrictions on the viewer of the page.
- You have an HTML template in a subdirectory of /windchill/codebase/ that will be used to generate the HTML page.

The post-condition is that an HTML page is generated presenting the desired information.

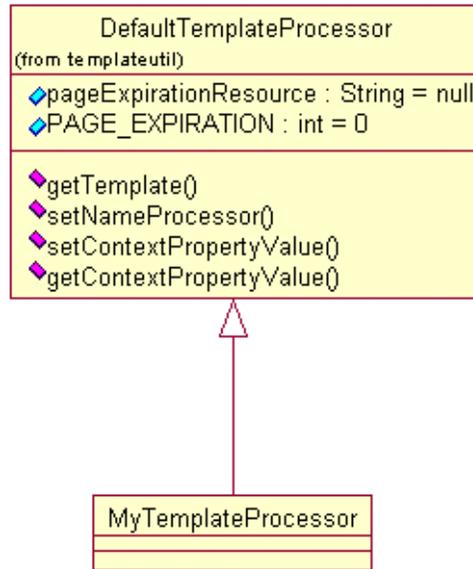
The default implementation steps are as follows:

1. Subclass `wt.templateutil.DefaultTemplateProcessor`. You can do this in either of the following two ways:

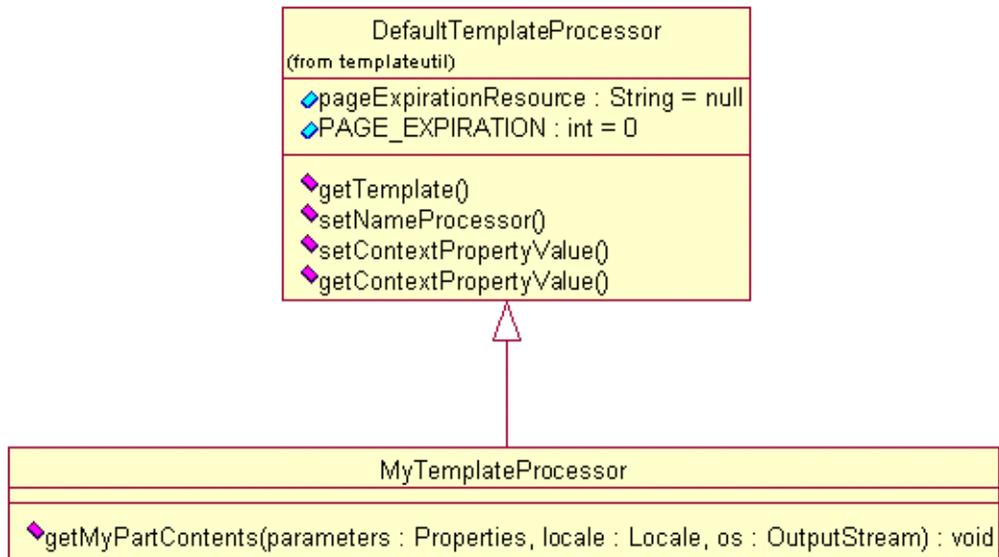
- Create the file in the editor of your choice and code your new class to extend `wt.templateutil.DefaultTemplateProcessor`:

```
import wt.templateutil.DefaultTemplateProcessor;  
public class MyTemplateProcessor extends DefaultTemplateProcessor
```

- Use Rational Rose to model your extension:



2. Add your specific Windchill script API calls that your template processor needs to support processing of your HTML templates. Again, you can do this in either of the following two ways:
  - Place all of the code in the class by editing it, using your source code editor of choice.
  - Add the API calls to your Rose model and generate the stub to be filled out:



**Note:** Displaying of the method signature is enabled to demonstrate the correct signature for a Windchill script API call.

3. Add a property so that TemplateProcessorFactory can find your new template processor using the guidelines in the section, Properties and Property Files.

```

wt.services/svc/default/wt.enterprise.TemplateProcessor/
<(Context Action)/<Context Class>/0=
<Template Processor Class Path>/duplicate
  
```

Variable fields have the following meanings:

**Context Action**

The value of the action parameter passed in either the query string or the Post data.

### Context Class

There are three possible values

- If an oid parameter is passed in, the Context Class is the class of the object represented by the oid.
- If no oid parameter is passed in but a class parameter is passed in, the class parameter is used to define the Context Class.
- If neither an oid nor class parameter is passed in the query string or POST data, java.lang.Object is used as the default Context Class.

The Context Class also follows the inheritance tree. That is why java.lang.Object will capture all concrete classes.

### Template Processor Class Path

The fully qualified path of the template processor subclass.

4. Add a property so that the DefaultHTMLTemplateFactory can find your new template processor using the guidelines in the section, Properties and Property Files.

Your property should have the following syntax:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/  
<Context Action>/<Context Class>/  
0=<(Relative Path from Codebase)>
```

The meanings of the first two variable fields, Context Action and Context Class, are the same as shown in the preceding step. The third field, *Relative Path from Codebase*, identifies the file path of the HTML template. In this path, you must replace the standard directory separators with periods ("."). For example, if you have an HTML template located at the following location on an NT system:

```
C:\Windchill\codebase\templates\myTemplates\MyTemplate.html
```

you would use the following entry for the relative path in the properties files:

```
...0= templates.myTemplates.MyTemplate...
```

5. Add view-based links to the navigation bar so your new page can be opened using the following steps:

- a. Open the file wt.enterprise.UriLinkResource.java.
- b. You must have a static String that is a reference to your business object. For purposes of this example, assume that MyPart is the business object. The entry should look like the following:

```
public final static String MYPART_ACTIONS = "11";
```

The String is composed by the rule: <Class Name>\_ACTIONS.

- c. Include a line that looks like the following:

```
{ MYPART_ACTIONS ,  
  "ObjProps:Properties,part_  
  MyPartContents:Contents of MyPart" },
```

The entries are of the following form:

<contextAction>:<Presentation Name of Link in Page>

The contextAction is value of the "action" that must be passed in either the query string or in the HTTP POST.

## Availability of the Page Depends on State or Permissions

This example describes how to create an HTML page that is a view of information for a specific business object, but whose availability depends on either the state of the object, the permissions of the viewer, or both.

The following are pre-conditions for the example:

- You need to present only a valid URL/PostData to set the context and the information for generating the view.

All template processing is required to have an action passed in either the QueryString or the HTTP POST data. The OID is used by the wt.fc.ReferenceFactory to retrieve the specific business object from persistent storage. You do not need any other information sent to generate the presentation of your page.

- The display of this page is contingent on the following conditions:
  - The state of the object
  - The access permissions of the viewer
  - Both of the preceding conditions
- You have an HTML template in a subdirectory of /windchill/codebase/ that will be used to generate the HTML page.

The post-condition is that an HTML page is generated presenting the desired information.

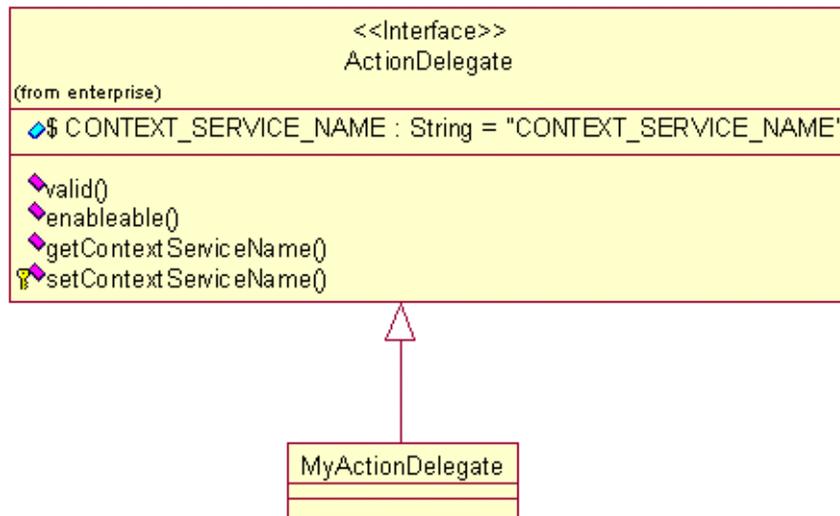
The implementation steps for this example follow. The first four steps are the same as those described in the preceding default implementation example; refer to that example for detailed instructions.

1. Subclass wt.templateutil.DefaultTemplateProcessor.
2. Add your specific Windchill script API calls that your template processor needs to support processing of your HTML templates.
3. Add a property so that TemplateProcessorFactory can find your new template processor.

4. Add a property so that DefaultHTMLTemplateFactory can find your new template processor.
5. Add permission/state dependent links to the navigation bar. This process involves several steps and requires use of the ActionDelegate/ActionDelegateFactory pair and the URLActionDelegate/URLActionDelegateFactory pair.
  - a. Decide if your action maps to one of the existing subclasses of wt.enterprise.ActionDelegate. Go to the Rose model and view the class diagram /wt/enterprise/Action to see all of the subclasses of wt.enterprise.ActionDelegate that are currently modeled.

The phrase *maps to* means that the functionality being checked by the subclass of ActionDelegate matches the action/view that you are presenting. For example, if you are presenting a link to check out your business class, then you should use the existing class, wt.enterprise.CheckOutActionDelegate. Otherwise, you need to subclass wt.enterprise.ActionDelegate yourself. Use the Rose model so the full collection of ActionDelegate subclasses is visible.

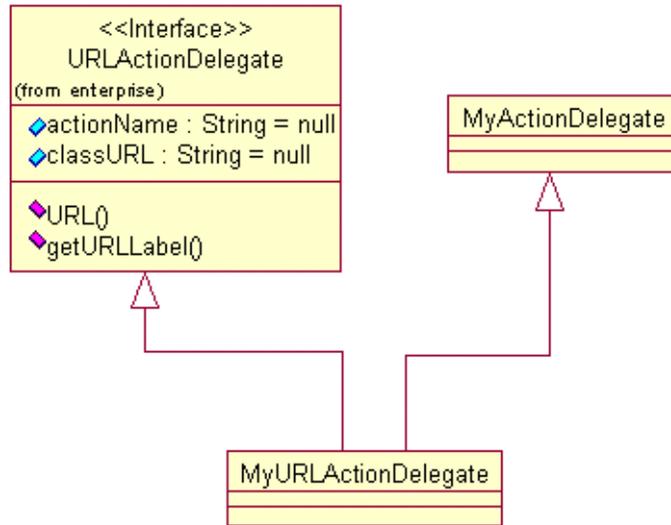
- b. If you need to create your own subclass of `wt.enterprise.ActionDelegate`, open the Rose model and model a new subclass of `wt.enterprise.ActionDelegate` and generate the stub:



- c. Fill in the method `valid( Object object )`. The purpose of this method is to check if the given action is valid on the current class.
- d. Fill in the method `enableable( Object object )`. The purpose of this method is to check if the given action is currently valid on the given object.
- e. Fill in a value for the `CONTEXT_SERVICE_NAME` that is unique to your subclass of `ActionDelegate`.
- f. Fill in the method `getServiceName` to return `CONTEXT_SERVICE_NAME`. That is, the single line in this method should be as follows:

```
return CONTEXT_SERVICE_NAME;
```

- g. Create a class that implements the `wt.enterprise.URLActionDelegate` interface that also subclasses the new `MyActionDelegate` that you just created:



- h. Fill in the stub method `URL( Object object )`. First, make a call to the `enable()` method in the parent `ActionDelegate` class, for example:

```

Boolean validation = enableable(object
if (validation == null || !validation.booleanValue
{
    throw new WTEException(RESOURCE,
        enterpriseResource.ERROR_INITIALIZING,param);
}

```

Then generate a `String` that presents a complete URL for that View/Action.

- i. Fill in the stub method `getURLLabel( Locale locale )`. This generates the text presented with the link in the generated page.
- j. Open the source for `wt.enterprise.URLLinkResource.java`.
- k. Add or update a line in `wt.enterprise.URLLinkResource.java` that looks like the following:

```

{ "MYWTPART_ACTION_LINKS",
AllVersionsActionDelegate.CONTEXT_SERVICE_NAME + "," +
VersionHistoryActionDelegate.CONTEXT_SERVICE_NAME + "," +
IterationHistoryActionDelegate.CONTEXT_
SERVICE_NAME + "," +
wt.lifecycle.LifeCycleHistoryActionDelegate.CONTEXT_SERVICE_
NAME + "," +
wt.lifecycle.SubmitActionDelegate.CONTEXT_SERVICE_NAME },

```

1. Add properties so that the ActionDelegateFactory and the URLActionDelegateFactory can find your new delegates. The following are two sample entries where CONTEXT\_SERVICE\_NAME refers to the value assigned to the Class variable -- CONTEXT\_SERVICE\_NAME -- in your ActionDelegate subclass:

```
wt.services/svc/default/wt.enterprise.ActionDelegate/  
<CONTEXT_SERVICE_NAME>/java.lang.Object/  
<ActionDelegate subclass>/singleton
```

```
wt.services/svc/default/wt.enterprise.URLActionDelegate/  
<CONTEXT_SERVICE_NAME>/java.lang.Object/  
0/<URLActionDelegate subclass>/singleton
```

## A View Page Requiring Custom Processing of QueryString or POST Data

This example describes how to create an HTML page that is a view of information for a specific business object, with no restrictions on viewing based on the state of the business object or the access permissions of the viewer. However, there is data in either the QueryString or in the Form data that requires initial processing to set the state of the template processor before beginning to process the HTML template.

Examples of such cases are as follows:

- You may have sent several auxiliary OIDs that require the persisted objects be retrieved before rendering the HTML page.
- There might be some state variables that define the presentation view that were sent and the view structure must be set before processing can begin.
- The current context object is not the object that will be used to process the HTML page. The true object to be used as the context object is navigable from the current context object.

The following are pre-conditions for this example:

- You need only present a valid URL/PostData to set the context and the information for generating the view.

All template processing is required to have an action passed in either the QueryString or the HTTP POST data. The OID is used by the wt.fc.ReferenceFactory to retrieve the specific business object from persistent storage. You do not need any other information sent to generate the presentation of your page.

- There are no contingencies on the state of the business object (for example, it need not be checked in or submitted) to present the view.
- There are no access restrictions on the viewer of the page.
- You have an HTML template in a subdirectory of /windchill/codebase/templates that will be used to generate the HTML page.

- You have information in the name/value pairs that arrive in the query string or POST data that must be processed before the processing of the HTML template can begin.

The post-condition is that an HTML page is generated presenting the desired information.

The implementation steps for this example are as follows. The first five steps are the same as those described in the default implementation example shown earlier in this section; refer to that example for detailed instructions.

1. Subclass `wt.templateutil.DefaultTemplateProcessor`.
2. Add your specific Windchill script API calls that your Template Processor needs to support for processing your HTML templates.
3. Add a property to `service.properties` so that `TemplateProcessorFactory` can find your new template processor.
4. Add a property in `htmltemplate.properties` so that `DefaultHTMLTemplateFactory` can find your new template processor.
5. Add view-based links to the navigation bar so your new page can be opened.
6. Override `readContext( )` and add instance fields.

The method `readContext( )` is responsible for setting the context of the HTML template process (that is, `contextAction`, `contextObj`, and `contextClassName`) and performing any other processing of the reading of the query string and/or POST data.

The POST data is available with a call to `getFormData()` and the query string data is available with a call to `getQueryString()`. Note that these method calls actually retrieve this data from the aggregated instance of `HTTPState` for which the template processor is a proxy.

The initial information and context that was used and set in the `URLProcessor` gateway is available through the `HTTPState` object. For example, if an object was retrieved from persistent storage based on an `OID` that was passed in either the query string or Form Data, that persistent object based on the `OID` is available from the `HTTPState` object with a call to `getContextObj()`. The action that was passed in is available from the call `getContextAction()`. Again, you will not reference the `HTTPState` object directly because the template processor is a proxy for the `HTTPState` object. You will call methods in your template processor that proxy for the `HTTPState` object.

## A View Page Requiring Custom Processing of HTML Template File Name or Path

This example describes how to create an HTML page that is a view of information for a specific business object, with no restrictions on viewing the page based on the state of the business object or the access permissions of the viewer. However, the implementation of the `DefaultHTMLTemplateFactory` is not sufficient or desirable in the sense of how it processes the property entry mapping a context action/object/class to a template.

Examples of such cases are as follows:

- You might have a number of pages that reside in the same directory and, instead of writing relative paths for each of the file names, you want to enter only the file name. You want to have the same relative path prepended to the file name for each file. Thus, you could easily switch directories. You could also use an entry in `wt.properties` to specify the relative path, allowing complete customizability.
- The entry might be a key that is mapped to another file. This type of implementation might be application to using `JavaHelp`. There might be other reason to use keys, such as when the information needs additional processing that is state-dependent or the information is persisted.

The following are pre-conditions for this example:

- You need only present a valid URL/PostData to set the context and the information for generating the view.

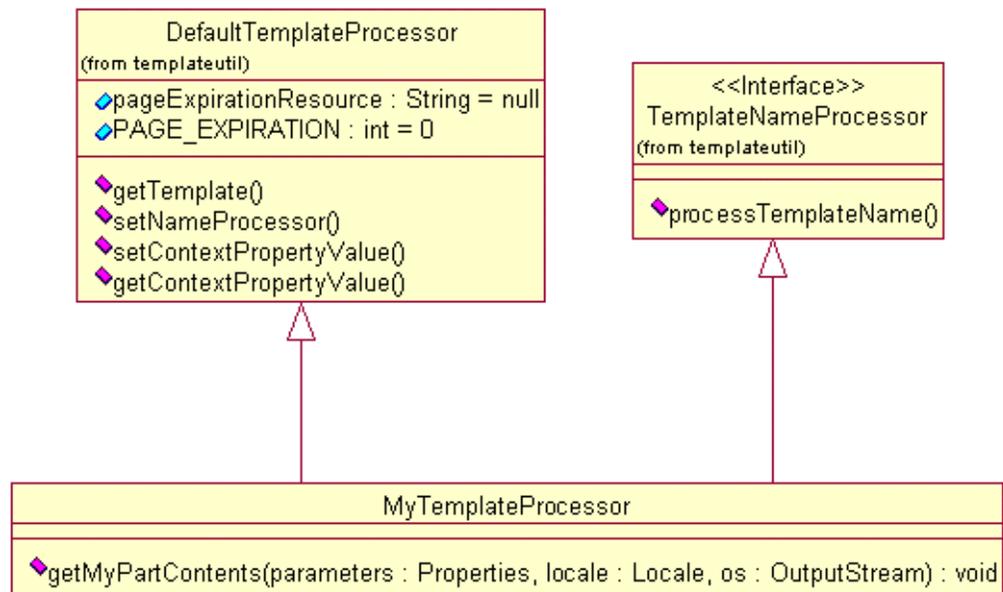
All template processing is required to have an action passed in either the `QueryString` or the HTTP POST data. The `OID` is used by the `wt.fc.ReferenceFactory` to retrieve the specific business object from persistent storage. You do not need any other information sent to generate the presentation of your page.

- There are no contingencies on the state of the business object (for example, it need not be checked in or submitted) to present the view.
- There are no access restrictions on the viewer of the page.
- You have an HTML template in a subdirectory of `/windchill/codebase/` that will be used to generate the HTML page.
- You require some additional processing of the relative path of the HTML template that cannot be incorporated into the `htmltemplate.properties` file or it is advantageous not to have the relative path entered.

The post-condition is that an HTML page is generated presenting the desired information.

The implementation steps for this example are as follows. The first five steps are the same as those described in the default implementation example shown earlier in this section; refer to that example for detailed instructions.

1. Subclass `wt.templateutil.DefaultTemplateProcessor`.
2. Add your specific Windchill script API calls that your Template Processor needs to support for processing your HTML templates.
3. Add a property so that `TemplateProcessorFactory` can find your new template processor.
4. Add a property so that `DefaultHTMLTemplateFactory` can find your new template processor.
5. Add view-based links to the navigation bar so your new page can be opened.
6. Implement the `wt.templateutil.TemplateNameProcessor` interface. There are two ways to implement this interface; they depend on whether you want the processing of the name to be available to other subclasses of `DefaultTemplateProcessor`.
  - If you only need the special processing of the entry in `htmltemplate.properties` in the one subclass of the `DefaultTemplateProcessor`, perform the following steps:
    - i. Have your subclass of `DefaultTemplateProcessor` implement the `TemplateNameProcessor` interface:



- ii. Put the following code in the no-arg constructor of your subclass of `DefaultTemplateProcessor`:

```
getHtmlTemplateFactory().setNameProcessor( this );
```

- If you want custom processing of the entry returned from `htmltemplate.properties` available to many template processors, perform the following steps:
  - i. Have a separate class implement the `TemplateNameProcessor` interface (for example, `MyTemplateNameProcessor`), to make service available to other template processors.
  - ii. Put the following code in the no-arg constructor of your subclass of `DefaultTemplateProcessor`:

```
getHtmlTemplateFactory().setNameProcessor( new
    MyTemplateNameProcessor() );
```

7. Update the `TemplateNameProcessor` instance in the `DefaultHTMLTemplateFactory` instance.

## Supplementary Details of Implementation

### Processing the QueryString/POST Data and Setting the Context

Within the `URLProcessor` gateway methods is an `HTTPState` object that is instantiated, initialized, and passed to the `TemplateProcessor`. The `DefaultTemplateProcessor` automatically acquires this `HTTPState` object and uses it. All of the `URLProcessor` gateway methods read both the `QueryString` and `Form Data` (if any is sent). Only the `processForm` gateway method can read an `HTTP POST` request with `enctype` set to `multipart/form-data`. The `URLProcessor` gateway method passes the `HTTPState` object into the `TemplateProcessor` on the `bizData` field of the `wt.httpgw.HTTPRequest` object in the `handleRequest` method.

The reading of the `URL/POST` data and the setting of the context is done in the `readContextStatic()` method in `URLProcessor`. This is where the values in the `HTTPState` object are set initially.

The `URLProcessor` requires that a valid `URL/Form` data must have at least an action parameter. If the action is missing, an exception is thrown. So the following name/value pair must be either in the `queryString` or the `Form Data`:

```
action=<Some Action>
```

A more common action is to include either an `OID` or a class parameter with the action parameter, as shown in the following examples:

```
action=<Some Action>&oid=<Some oid generated by ReferenceFactory>
```

```
action=<Some Action>&class=<The name of a class>
```

If neither an `oid` nor a class are on the `URL` or in the `POST` data, the factories will use an instance of `java.lang.Object` as the default context object.

The default parsing of the URL/POST data results in the variable `contextAction` in an `HTTPState` object being set to the value of the action passed in. If an oid is read, the `contextObj` in the same `HTTPState` object is set using a call to `ReferenceFactory` to get the persisted object. If a class value is sent and there is not an oid, then the `String` `className` in the same `HTTPState` object is set to the `String` that is sent.

All of the original values from the `QueryString` are passed into the `queryData` field in the `HTTPState` object. All of the original name/values pairs from the Form Data are placed in the `formData` field in the `HTTPState` object. When the `HTTPState` object is passed into the `TemplateProcessor`, all of these values are immediately available.

## Finding and Initializing the HTML Template to be Processed

`DefaultTemplateProcessor` uses an aggregated instance of `wt.templateutil.DefaultHTMLTemplateFactory` to locate and initialize the HTML template file to process. The aggregated instance of `DefaultHTMLTemplateFactory` is used with its default settings. The usage and the default settings of `DefaultHTMLTemplateFactory` are described in the section on Using the HTML Template Factory later in this chapter. Using the context that was set by parsing the URL, the context in the `DefaultHTMLTemplateFactory` is set with the following code:

```
DefaultHTMLTemplateFactory htmlTemplateFactory =
    getHtmlTemplateFactory();

htmlTemplateFactory.setState( getState() );
htmlTemplateFactory.setLanguagePreferences( req );
```

The implementation of `DefaultHTMLTemplateFactory` uses the Context Object if it is not null; otherwise, the Context Class Name is used to find the desired entry in `htmltemplate.property`.

Using the default implementation in `DefaultTemplateProcessor` and the default implementation in `DefaultHTMLTemplateFactory`, the following line is retrieved from `htmltemplate.properties`:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
<Context Action>/<Context Object or Context Class>/
0=<Your desired entry>
```

The `String` "Your desired entry" is returned. The default implementation will have all occurrences of the period (".") replaced by the slash character ("/") in the `String` "Your desired entry".

**Note:** The `String` that results after this replacement must represent a relative path from `/codebase/` to some HTML template file.

## Windchill Script API Supported

DefaultTemplateProcessor subclasses wt.enterprise.BasicTemplateProcessor. Therefore, all of the Windchill script API calls supported in BasicTemplateProcessor are available. DefaultTemplateProcessor does not implement any Windchill script API calls itself. Therefore, only those in BasicTemplateProcessor are available by default.

## Available Parameters

The following parameters are available:

### **PAGE\_EXPIRATION**

Specifies in milliseconds the expiration time of the HTML page that is generated. There are set/get methods for this parameter.

### **PageExpirationResource**

String that represents a name in the wt.properties file. If you set this parameter, DefaultTemplateProcessor looks for the value of this parameter in wt.properties and, if found, uses it to set the page expiration time.

# How to Use the HTML Template Factory

This section describes how to use the HTML template factory service as a standalone service. (It does not discuss in detail the usage of the HTML template factory service in template processors or the `wt.templateutil.DefaultTemplateProcessor` in particular.)

## Overview

The following code represents the default usage of the HTML template factory. The comments describe the activity represented by the method call that follows.

```
// The initial construction of the default implementation
// of the AbstractHTMLTemplateFactory
DefaultHTMLTemplateFactory defaultHTMLTemplateFactory =
    getHtmlTemplateFactoryStatic();

// Setting the context of the HTTP request.
defaultHTMLTemplateFactory.setContextAction( "ObjProps" );
defaultHTMLTemplateFactory.setContextObj( null );
defaultHTMLTemplateFactory.setContextClassName(
    "wt.doc.WTDocument" );

// The call that will be used to set the information
// required initialize and localize the HTMLTemplate.
// The object, request is of class wt.httpgw.HTTPRequest
defaultHTMLTemplateFactory.setLanguagePreferences( request );

// The call that returns an initialized HTMLTemplate object
HTMLTemplate template =
    defaultHTMLTemplateFactory.getHTMLTemplate();
```

`DefaultHTMLTemplateFactory` performs the following process when the method `getHTMLTemplate()` is invoked:

1. Based on the context that you set (Action/Object or Action/ClassName pair), the `DefaultHTMLTemplateFactory` looks in the application context service properties for an entry with the following format:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
    <ContextAction>/<ContextObject>/0=<Return String>
```

2. The current aggregated instance of the `TemplateNameProcessor` interface is called to process the property entry and return a modified String. In the default case, any period (".") in the entry is replaced with system-based file separators. For example, `templates.forms.CreateWTDocument` becomes `templates/forms/CreateWTDocument`.
3. The modified String is used as a relative path name from the `/windchill/codebase/` directory to locate the `HTMLTemplate` object.
4. Using the Language preference set by the calling class, the `HTMLTemplate` is localized.
5. The localized `HTMLTemplate` object is initialized and returned.

The major steps necessary to use the default implementation of the HTML template factory service are as follows:

1. Create an instance of `DefaultHTMLTemplateFactory`.
2. Set the context (described in greater detail in the section on Default Usage later in this section).
3. Pass in the information required to initialize and localize the `HTMLTemplate` object (described in greater detail in the section on Setting the Locale Preference later in this section).
4. Call the `getHTMLTemplate()` method.

To customize the HTML template factory service, you can take either of the three following actions:

- Implement your own subclass of `wt.templateutil.AbstractHTMLTemplateFactory`.
- Implement your own subclass of `wt.templateutil.DefaultHTMLTemplateFactory`.
- Use one of the customization points of the `DefaultHTMLTemplateFactory` (see the sections on overriding the entry and the default setting in `htmltemplate.properties` later in this section).

## Default Usage

The standard usage of `DefaultHTMLTemplateFactory` is represented in the following examples. These examples cover the three contexts that can occur when retrieving an HTML template. In these examples, the request object is an instance of a `wt.httpgw.HTTPRequest` object.

## Context Action and Context Object

The following example has a context action and a context object:

```
DefaultHTMLTemplateFactory defaultHTMLTemplateFactory =
    new DefaultHTMLTemplateFactory ();
defaultHTMLTemplateFactory.setContextAction( "ObjProps" );
defaultHTMLTemplateFactory.setContextObj( getContextObj() );
    // Current Object is a WTPart
defaultHTMLTemplateFactory.setLanguagePreferences( request );

HTMLTemplate template =
    defaultHTMLTemplateFactory.getHTMLTemplate();
```

You would have to create the following application context service property, mapping the action and object to a template:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
ObjProps/wt.part.WTPart/
0=templates.myProperties.myPartPropertiesPage
```

The following HTML template file (or the correctly localized version of it) would be searched for:

```
Windchill/codebase/templates/myProperties/  
myPartPropertiesPage.html
```

By default, the periods (".") in an entry in the `htmltemplate.properties` file is replaced with a slash ("/") and then that String is used as a relative path to the HTML template file.

## Context Action and Context Class Name

The following example has a context action and a context class:

```
DefaultHTMLTemplateFactory defaultHTMLTemplateFactory =  
    new DefaultHTMLTemplateFactory ();  
defaultHTMLTemplateFactory.setContextAction( "ObjProps" );  
defaultHTMLTemplateFactory.setContextClassName(  
    "wt.doc.WTDocument" );  
defaultHTMLTemplateFactory.setLanguagePreferences( request );  
  
HTMLTemplate template =  
    defaultHTMLTemplateFactory.getHTMLTemplate();
```

You would have to create the following application context service property, mapping the action and object to a template:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/  
ObjProps/wt.doc.WTDocument/  
0=templates.myProperties.myDocumentPropertiesPage
```

The following HTML template file (or the correctly localized version of it) would be searched for:

```
Windchill/codebase/templates/myProperties/  
myDocumentPropertiesPage.html
```

By default, a period (".") in an entry in the `htmltemplate.properties` file is replaced with a slash ("/") and then that String is used as a relative path to the HTML template file.

## Context Action Only

The following example has a context action only:

```
DefaultHTMLTemplateFactory defaultHTMLTemplateFactory =  
    new DefaultHTMLTemplateFactory ();  
defaultHTMLTemplateFactory.setContextAction( "ObjProps" );  
defaultHTMLTemplateFactory.setLanguagePreferences( request );  
  
HTMLTemplate template =  
    defaultHTMLTemplateFactory.getHTMLTemplate();
```

Note that no `ContextObject` or `ContextClassName` is set. When neither is set, the default response by the `DefaultHTMLTemplateFactory` is to use a `java.lang.Object` object to look for a property mapping the action to a template..

Your property would look as follows:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/  
  ObjProps/java.lang.Object/  
  0=templates.myProperties.mySimplePropertiesPage
```

The following HTML template file (or the correctly localized version of it) would be searched for:

```
Windchill/codebase/templates/myProperties/  
  mySimplePropertiesPage.html
```

By default, a period (".") in the property is replaced with a slash ("/") and then that String is used as a relative path to the HTML template file.

## The wt.templateutil.ContextHolder Interface

In this example, the calling class implements the wt.templateutil.ContextHolder interface (similar to wt.enterprise.BasicTemplateProcessor, for example):

```
DefaultHTMLTemplateFactory defaultHTMLTemplateFactory =  
  new DefaultHTMLTemplateFactory ();  
defaultHTMLTemplateFactory.copyContextFrom( this ); // calling  
  // class implements ContextHolder interface  
defaultHTMLTemplateFactory.setLanguagePreferences( request );  
  
HTMLTemplate template =  
  defaultHTMLTemplateFactory.getHTMLTemplate();
```

The call to copyContextFrom results in the ContextAction, the ContextObj, and the ContextClassName objects being copied from the ContextHolder implementation that is passed in the call into the instance of DefaultHTMLTemplateFactory. Make sure that your context is current when using this method. You must also know the logic used by DefaultHTMLTemplateFactory to select the HTML template in this case. The contextAction that is passed in is used directly. If there is a context object, this is used directly. If there is not a context object but there is a context class name, this is used. If there is not a context object or a context class name, then an instance of java.lang.Object is used for the context object.

You would have to create a property such as the following:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/  
  ObjProps/java.lang.Object/  
  0= templates.myProperties.mySimplePropertiesPage
```

The following HTML template file (or the correctly localized version of it) would be searched for:

```
Windchill/codebase/templates/myProperties/  
  mySimplePropertiesPage.html
```

By default, a period (".") in the property is replaced with a slash ("/") and then that String is used as a relative path to the HTML template file.

## Setting the Locale Preference

To get a properly localized HTML template, the correct Locale must be set. To pass in the sufficient information to `DefaultHTMLTemplate` (any subclass of `AbstractHTMLTemplate`, in fact), you must call one of the following APIs:

- `defaultHTMLTemplateFactory.setLocale(locale)` if you have a locale available
- `defaultHTMLTemplateFactory.setLanguagePreferences(request)` if you have an `HttpServletRequest` object
- `defaultHTMLTemplateFactory.setLanguagePreferences(Vector)` if you have a `Vector` that is the result of the call `LanguagePreference.getAcceptLanguagePreferences(String)`

## Overriding the `htmltemplate.properties` Entry

The property with the service name, `wt.templateutil.defaultHTMLtemplate` is used to define the relative path to the HTML template file. By default, any period (".") in the property is replaced with a slash ("/"). The resulting String must be a relative path to the desired HTML template file.

To implement your own processing of the String returned from the property, you must implement the `wt.templateutil.TemplateNameProcessor` interface. A concrete instance of this interface is used by `DefaultHTMLTemplateFactory` to replace the period (".") in the String with the slash ("/"). Once you implement this interface, you pass it into `DefaultHTMLTemplateFactory`, before calling `getHTMLTemplate()`, with the following call:

```
defaultHTMLTemplateFactory.setNameProcessor(  
    myNameProcessor );
```

This interface performs the following actions:

- It receives the template name for a property with the application context service name, `wt.templateutil.defaultHTMLtemplate`.
- It processes the name in a way that suits the developer's need.
- It returns a String that must represent a relative class path to an HTML template file.

Simple cases where the developer might want to override the default response are as follows:

- When all the files of a certain type always reside in the same directory. In this case, there is no need to enter the complete relative path. This allows freedom in the future to move the directory without having to edit many different entries.

- When the template name in the property is actually a key and then the implementation of the `wt.templateutil.TemplateNameProcessor` interface knows how to interpret the key.

## Overriding the Service Name in `htmltemplate.properties`

The fourth element in each application context service property is the service name. For example, in the following entry, the service name is `wt.templateutil.DefaultHTMLTemplate`:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/  
ObjProps/wt.part.WTPart/0=  
templates.myProperties.myPartPropertiesPage
```

To select a different service name, use the API `setServiceName()`.

## Adding an Action in the HTML Client—Example 2

This example shows how to create and add a new page with an HTML form to the HTML client. The form will update a property on a document. Then the response page will display the newly updated document.

The objectives of this exercise are as follows:

- To describe the basic principles of adding an action page to the HTML client, showing you the most important classes to create, and the property files to modify.
- To describe how to process an HTML form.
- To learn how to set and generate the response page.

The four major steps of this example are as follows:

1. Add a state/permission dependent link.
2. Create a subclass of `wt.templateutil.processor.GenerateFormProcessor` that supports the calls within the new HTML template to generate an HTML form.
3. Create a subclass of `wt.templateutil.processor.FormTaskDelegate` to process an HTTP POST and specify the response page.
4. Generate the response page.

The example actually consists of two major parts: first, creating the update page and adding a link to the update page and, second, processing the HTML form and sending a response.

### Before Starting

Before starting this example, open your `wt.properties` file and add the following line as the last entry:

```
wt.template.cache.enabled=false
```

Save and close the file.

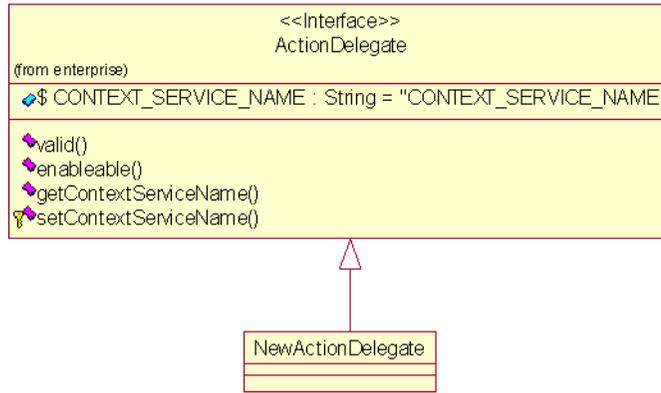
**Note:** This property should be set to "false" for development, but would normally be "true" in production mode.

### Creating the ActionDelegate and URLActionDelegate

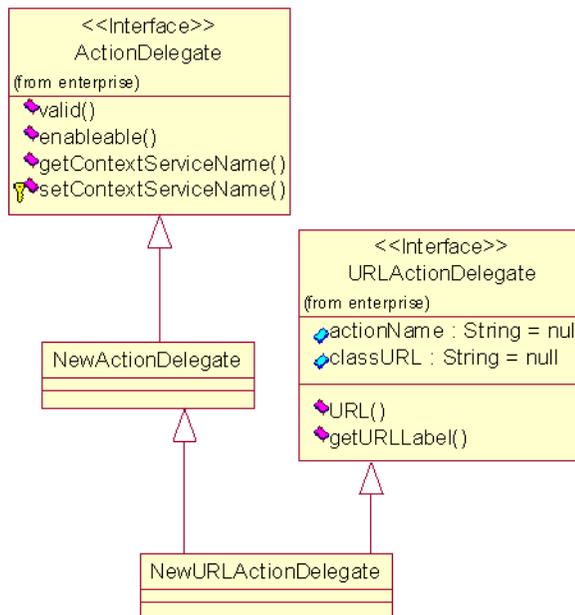
To create the `ActionDelegate` and `URLActionDelegate`, perform the following steps:

1. Open Rational Rose and load `/src/WTdesigner.mdl`.
2. Go to the `training.clientlab` package.

3. Add a new class diagram to that package and name the new diagram Add Action.
4. Add a class to the Add Action class diagram and name it ActionDelegate.
5. Add a class to the Add Action class diagram and name it NewActionDelegate. Have the NewActionDelegate class subclass the ActionDelegate class.



6. Add a Class to the Add Action class diagram and name it URLActionDelegate.
7. Add a class to the Add Action class diagram and name it NewURLActionDelegate. Have the NewURLActionDelegate subclass both the NewActionDelegate class and the URLActionDelegate. Note that for readability, the fields in the ActionDelegate class have been suppressed.



8. Generate the code.

9. In `NewActionDelegate`, fill in the valid method stub with the following code (also found in `customization/HTMLClient/Example2.doc` under the heading "valid Method"):

```
return new Boolean( ( (object instanceof wt.vc.wip.Workable) &&
    (object instanceof wt.doc.WTDocument) ) );
```

This method determines whether the current action is valid on the current object type.

10. In `NewActionDelegate`, fill in the enable method stub with the following code (also found in `customization/HTMLClient/Example2.doc` under the heading "enable Method"):

```
if( object == null )
    {
        throw new WTEException( null, "wt.clients.doc.DocRB",
            wt.clients.doc.DocRB.NO_DOCUMENT_TO_UPDATE, null );
    }
    if (!(valid(object).booleanValue())) { //is it Workable

        throw new WTEException("Object is either not a document
            or not workable");
    }

    if (!(wt.access.AccessControlHelper.manager.hasAccess(
        object,wt.access.WTPermission.MODIFY))) {
        String action = wt.util.WTMessage.getLocalizedMessage(
            "wt.enterprise.enterpriseResource",

wt.enterprise.enterpriseResource.MODIFY,
            null);
        Object[] param = { action };
        throw new WTEException("wt.enterprise.enterpriseResource",
            wt.enterprise.enterpriseResource.NOT_PERMITTED,
            param);
    }

    if (!( //is the document checkout or user is admin
        (wt.vc.wip.WorkInProgressHelper.isCheckedOut((
            wt.vc.wip.Workable)object,
            wt.session.SessionMgr.getPrincipal())) ||
        (
            (wt.access.AccessControlHelper.manager.hasAccess(object,
                wt.access.WTPermission.ADMINISTRATIVE)) &&&
            (
                (wt.vc.wip.WorkInProgressHelper.isCheckedOut((
                    wt.vc.wip.Workable)object)) &&&
                (!wt.vc.wip.WorkInProgressHelper.isCheckedOut((
                    wt.vc.wip.Workable)object,
                    wt.session.SessionMgr.getPrincipal())))))
            {
                Object[] param = {
                    ((wt.fc.WTObject)object).getDisplayIdentity() };
                throw new WTEException("wt.enterprise.enterpriseResource",
                    wt.enterprise.enterpriseResource.CHECKIN_FAILED,
```

```

        param);
    }

    return new Boolean( true );

```

The enable method determines if the current action should be enabled on the current object based on the current object's state and the current user's permission.

11. In NewActionDelegate, add the following field (also found in customization/HTMLClient/Example2.doc under NewActionDelegate Field) in the Rose-generated user.attributes section:

```
public static final String CONTEXT_SERVICE_NAME = "NEWACTION";
```

Select a handle for the new ActionDelegate by replacing <DelegateHandle> with a name composed of uppercase letters.

The getContextServiceName method returns the handle, or name, of the current ActionDelegate.

12. In NewActionDelegate, fill in the getContextServiceName method stub with the following code (found also in customization/HTMLClient/Example2.doc under the getContextServiceName Method heading):

```
return CONTEXT_SERVICE_NAME;
```

13. In NewURLActionDelegate, fill in the URL method stub with the following code (also found in customization/HTMLClient/Example2.doc under the "URL Method"):

```

Object[] param = {this.getClass().getName ()};
Boolean validation = enableable(object);
if (validation == null || !validation.booleanValue())
{
    throw new WTEException("wt.enterprise.enterpriseResource",
        wt.enterprise.enterpriseResource.ERROR_INITIALIZING,
        param);
}

java.net.URL url = null;
try
{
    wt.vc.wip.Workable vcObj = ( wt.vc.wip.Workable )object;

    java.util.Properties props = new java.util.Properties();
    props.put("action", "&lt;FORM_ACTION>");

    wt.fc.ReferenceFactory rf = new wt.fc.ReferenceFactory( );
    wt.vc.VersionReference ref =
        wt.vc.VersionReference.newVersionReference( vcObj );
    String oid = rf.getReferenceString( ref );

    props.put("oid",oid);
    Class urlp = Class.forName("wt.enterprise.URLProcessor" );
    url = wt.httpgw.GatewayURL.buildAuthenticatedURL
        (urlp.getName(),"generateForm", props);
}

```

```

    }
    catch (WTEException wte)
    {
        throw new
WTEException(wte, "wt.enterprise.enterpriseResource",
            wt.enterprise.enterpriseResource.ERROR_INITIALIZING,
            param);
    }
    catch (java.lang.ClassNotFoundException cnfe)
    {
        throw new
WTEException(cnfe, "wt.enterprise.enterpriseResource",
            wt.enterprise.enterpriseResource.ERROR_INITIALIZING,
            param);
    }
    finally
    {
    }
    return url.toString();

```

Replace the entry <FORM\_ACTION> with a meaningful name; it need not be all uppercase or all lowercase.

The URL action checks for permission to display the link. If permission is granted by the parent ActionDelegate, the correct URL is returned. The String is placed on the URL and used to indicate which FormTaskDelegate to use.

14. In NewURLActionDelegate, choose a name for the hyperlink and have the getURLLabel method return this name as a String. For example, if you want the link in the page to read My New Action, then have getURLLabel return the String "My New Action" as :return "My New Action";
15. Compile the code.

## Registering the ActionDelegate and the URLActionDelegate

To register the ActionDelegate and the URLActionDelegate, perform the following steps:

1. Add the following application context service property for the ActionDelegate:

```

services/svc/default/wt.enterprise.ActionDelegate/
<DelegateHandle>/java.lang.Object/
0=clientlab.NewActionDelegate/singleton

```

Replace <DelegateHandle> with the name of the handle you gave your ActionDelegate in step 11 earlier in this example.

2. Add the following application context service property for the URLActionDelegate:

```
wt.services/svc/default/wt.enterprise.URLActionDelegate/  
  <DelegateHandle>/java.lang.Object/  
  0=clientlab.NewURLActionDelegate /singleton
```

You must replace <DelegateHandle> with the name of the handle you gave your URLActionDelegate in step 11 earlier in this example.

The ActionDelegateFactory and the URLActionDelegateFactory should now be able to find your new ActionDelegates in the <DelegateHandle>/<Class> that you specified in the new property.

See the section, Properties and Property Files for information on how to add new properties to the system.

## Adding a Link to the HTML Client for Your New Page

To add a link to the HTML client for your new page, perform the following steps:

1. Open the wt/enterprise/UrlLinkResource.java in your favorite editor.
2. Extend the WTDOCUMENT\_ACTION\_LINKS entry in the resource bundle by adding the following line (found also in customization/HTMLClient/Example2.doc under the UrlLinkResource Entry heading):

```
< ACTIONDELEGATE_CLASS>.CONTEXT_SERVICE_NAME + ", " +
```

Replace <ACTIONDELEGATE\_CLASS> with the fully qualified class path of the ActionDelegate subclass you created earlier in the example.

In practice, you will update all of the localized versions of UrlLinkResource.

3. Save the file and compile. UrlLinkResource now has a new link for Documents in its listing.

## Creating the HTML Template to Generate the HTML Form

To create the HTML template to generate the HTML form, perform the following steps:

1. Copy the following HTML (also found in customization/HTMLClient/Example2.doc under the heading HTML Form Template Skeleton) and paste it into a new HTML file. This will be your HTML template.

```
<HTML>
```

```
<SCRIPT LANGUAGE=Windchill>  
wt.htmlutil.HtmlUtil createBase  
</SCRIPT>
```

```
<HEAD>
```

```

    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=
      <SCRIPT LANGUAGE=Windchill>getCharsetEncoding</SCRIPT>">
<TITLE>Update Document</TITLE>
<STYLE TYPE="text/css">
  <!--
    FONT,H1,H2 {font-family:<SCRIPT LANGUAGE=
      Windchill>getWCFontFamily quotes=none</SCRIPT>}
  -->
</STYLE>
</HEAD>

<HTML>

<BODY BGCOLOR=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=bg-body</SCRIPT>
  TEXT=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=f-body</SCRIPT> >

<SCRIPT LANGUAGE=Windchill>
<!--
showResponseHeaders
-->
</SCRIPT>

<SCRIPT LANGUAGE=Windchill>
<!--
showResponseExceptions
-->
</SCRIPT>

</SCRIPT>

<FORM NAME="updateDocument" ACTION=
<SCRIPT LANGUAGE=Windchill>
getURLProcessorLink method=processForm action=<TASK_ACTION>
</SCRIPT>
METHOD = POST>

<TABLE BORDER=0 WIDTH=100%>

<TR ALIGN=LEFT>
  <TD ALIGN=RIGHT VALIGN=TOP WIDTH=15%
    BGCOLOR=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=
      bg-navbar</SCRIPT> CELLPADDING=0 CELLSPACING=0>

    <TABLE WIDTH=100% CELLPADDING=0 CELLSPACING=0>
      <TR ALIGN=RIGHT>
        <TD>
          <SCRIPT LANGUAGE=Windchill>
            <!--
              createGlobalNavigationBar
            -->
          </SCRIPT>
        </TD>
      </TR>
    </TABLE>
  </TD>
</TR>

<P>
<BR>

```

```

<TD ALIGN=LEFT VALIGN=TOP>

<TABLE WIDTH="500" CELLPADDING="0">

<TR><TD>
  <TABLE BORDER="0" CELLPADDING="1" WIDTH="500">
    <TR><TD ALIGN=LEFT BGCOLOR=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=
      bg-pagetitle</SCRIPT> HEIGHT="30">
      <B><FONT SIZE="3" COLOR=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=
        f-pagetitle</SCRIPT> >
      <H2>Update Document</H2>
    </TD></TR>
  </TABLE>
</TD></TR>

<TR><TD>

<TABLE BGCOLOR=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=
  bg-form</SCRIPT> BORDER="0"
  WIDTH="500" CELLPADDING="3">

  <TR>
    <TD ALIGN=RIGHT>
      <B><FONT>Department:
    </TD>
    <TD>
      <SCRIPT LANGUAGE=Windchill>
        enumeratedTypeMenu propertyName=department
      </SCRIPT>
    </TD>
  </TR>

  <TR>
    <TD ALIGN=RIGHT>
      <B><FONT>Title:
    </TD>
    <TD>
      <INPUT NAME = "title" TYPE="text" SIZE=50 VALUE="<SCRIPT LANGUAGE=
        Windchill>contextualValue propertyName=title</SCRIPT>">
    </TD>
  </TR>

</TABLE>

<INPUT TYPE=hidden name=oid VALUE=<SCRIPT LANGUAGE=Windchill>getOID</SCRIPT> >

<INPUT TYPE=submit VALUE=" Ok ">

</TD>
</TABLE>

</FORM>

```

```
<SCRIPT LANGUAGE=Windchill>
<!--
showResponseFooters
-->
</SCRIPT>
```

```
</HTML>
```

2. In the HTML template you just created containing the preceding code, locate the `<Form action=...` section and replace the `<TASK_ACTION>` entry (shown in bold in the preceding code) with some name relevant to the action you are performing on this page. The `<TASK_ACTION>` value will be used in the next stage of the process to locate the `FormTaskDelegate` that processes the HTML form.
3. Save the file under the codebase directory. You now have a new HTML template file.

## Registering the HTML Template

To register the HTML template, add the following new application context service property:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
  <FORM_ACTION>/wt.doc.WTDocument/0=<FORM_TEMPLATE_NAME>
```

Replace `<FORM_ACTION>`. The correct value comes from the value used to replace the `<FORM_ACTION>` entry in the `URLActionDelegate`'s `URL` method in step 13 of the example.

You must also fill in `<FORM_TEMPLATE_NAME>` with the relative path of the HTML template that you created in the previous action. You must use the period (`.`) as a directory separator and leave off the file extension. For example, the following path:

```
templates/ObjectProperties/WTPart.html
```

should be entered as follows:

```
templates.ObjectProperties.WTPart
```

See the section, `Properties and Property Files` for information on how to add new properties to the system.

## Creating the Template Processor

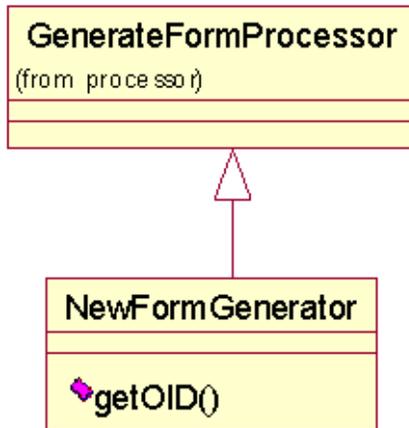
To create the template processor, perform the following steps:

1. Open Rational Rose and load `/src/WTdesigner.mdl`. You will be modeling your template processor as a subclass of `wt.templateutil.processor.DefaultTemplateProcessor`.

2. Add a new class to the Add Action diagram and name it GenerateFormProcessor.
3. Add a new class to the Add Action diagram and name it NewFormGenerator. Then have your new class subclass GenerateFormProcessor.
4. Add a new method to the NewFormGenerator class. The signature of the method is as follows:

**getOID(Properties parameters, Locale locale, OutputStream os)**

When you are done, your class diagram should look like the following:



5. Generate the training.clientlab package. You should now have a template processor, NewFormGenerator.java, in the package training.clientlab.
6. Fill in the method stub getOID with the following code (found also in customization/HTMLClient/Example2.doc under the heading getOID method):

```

java.io.PrintWriter out = getPrintWriter( os, locale )
try
{
    wt.fc.ReferenceFactory factory = new wt.fc.ReferenceFactory();
    String oid = factory.getReferenceString(
        (wt.fc.Persistable)getContextObj() );
    out.print( oid );
}
catch (wt.util.WTException wte)
{
    out.print(" ");
    addToResponseFooters(wte);
}
out.flush();
  
```

7. Compile the new Java file. You now have a new template processor.

## Registering the Template Processor

To register the template processor, add the following new application context service property:

```
wt.services/svc/default/wt.enterprise.TemplateProcessor
<FORM_ACTION>/ wt.doc.WTDocument/
0=clientlab.NewFormProcessor/duplicate
```

Replace <FORM\_ACTION> with the name of the <FORM\_ACTION> you selected in step 13 of creating the URLActionDelegate.

See the section, Properties and Property Files for information on how to add new properties to the system. At this point, a new link to the new HTML form page should exist.

## Finding Your New HTML Page

To verify that your new HTML page exists and can be accessed, perform the following steps:

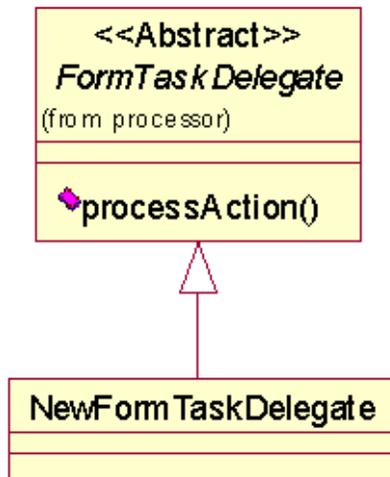
1. Stop and restart the method server and the server manager.
2. From the Windchill home page, go to the Local Search.
3. Perform a search on Documents.
4. When the results appear, click the number of any of the documents that are checked out. This should take you to the object properties page of the selected Document. UrlLinkResource now has a new link for Documents in its listing.
5. A link with the text that you entered in the URLActionDelegate display should exist in the second section of the Navigation bar. Click that link to return your new page with the HTML form in it.

The following steps form the second major part of the example, Processing the HTML Form and Sending a Response.

## Creating the FormTaskDelegate

To create the FormTaskDelegate, perform the following steps:

1. Open Rational Rose and load /src/WTdesigner.mdl. You will be modeling your template processor as a subclass of wt.templateutil.processor.DefaultTemplateProcessor.
2. Add a new class to the Add Action diagram and name it FormTaskDelegate.
3. Add a new class and name it NewFormTaskDelegate. Then have your new class subclass FormTaskDelegate. When you are done, your class diagram should look like the following:



4. Generate the training.clientlab package. You should now have a template processor, NewFormTaskDelegate.java, in the package training.clientlab.
5. Open NewFormTaskDelegate and paste in the following code (found also in customization/HTMLClient/Example2.doc under the heading performAction method) into the performAction code stub in NewFormTaskDelegate:

```
String title = getFormData().getProperty("title");

if (title == null || title.length() == 0)
{
    setContextAction("<ERROR_ACTION>");
    addToResponseHeaders(new wt.util.WTException(
        "You must enter a Title with length between 1 and 20
        characters"));
    return;
}

if (title != null && title.length() > 20)
{
    setContextAction("<ERROR_ACTION>");
    addToResponseHeaders(new wt.util.WTException(
        "Title too long. You must enter a Title with length
        between 1 and 20 characters"));
}
```

```
        return;  
    }  
  
    setContextAction( "<RESPONSE_ACTION>" );  
    return;  
}
```

6. Replace <ERROR\_ACTION> with the value you replaced <FORM\_ACTION> with in step 2 of Registering the HTML Template.

This defines the action that is performed if changing the attributes value fails. Generally, you want to have the original HTML form come back with a message indicating why the form failed.

7. Replace <RESPONSE\_ACTION> with the entry: "ObjProps". This entry defines the action that is to be taken in the case of a successful update.
8. Save and compile the new Java file. You now have a new NewFormTaskDelegate.

## Registering the FormTaskDelegate

To register the FormTaskDelegate, add the following new application context service property:

```
wt.services/svc/default/wt.templateutil.processor.  
FormTaskDelegate/<TASK_ACTION>/wt.doc.WTDocument/  
0=clientlab.NewFormTaskDelegate/duplicate
```

Replace <TASK\_ACTION> with the name of the <TASK\_ACTION> you selected in step 2 of Creating the HTML Template to Generate the HTML Form.

See the section, Properties and Property Files for information on how to add new properties to the system.

The FormTaskDelegateFactory should now be able to find your new FormTaskDelegate subclass based on the <TASK\_ACTION>/<Class> that you specified in the new entry.

At this point, you should be able to update the property of the document.

## Finding Your New HTML Page

To find your new HTML page, perform the following steps:

1. Stop and restart the method server and the server manager.
2. From the Windchill home page, go to the Local Search.
3. Perform a search on Documents.
4. When the results come back, click the number of any of the documents that are checked out. This should take you to the object properties page of the document. `UrlLinkResource` now has a new link for Documents in its listing.
5. A link with the text that you entered in the `URLActionDelegate` display should exist in the second section of the Navigation bar.
6. Click that link to return your new page with the HTML form in it.
7. Fill in a value of more than 20 letters and click **Submit**.
8. The form should come back with a message at the top indicating why the update failed. Repeat the preceding step with 20 or fewer letters.
9. The object properties page for the document should be returned.

## Form Processing and Action Processing

This section describes how to perform actions and process forms within the Windchill system through HTML pages. There are essentially two ways to do this:

- Perform an action that does not require any user input. The user clicks on a link and some action is performed in the Windchill system. An example of this is checking out a document.
- Perform an action that requires the user to fill out a form. After the form is completed, the user submits the form and some action takes place using the data in the form. An example of this is creating a document.

The remainder of this section describes the steps essential to implementing each of these cases, followed by examples.

### Overview of Invoking an Action

Following are the general steps that take place when an action is invoked:

1. A link is generated in an HTML page that represents a link to invoke an action in the Windchill system. If making the link available is dependent on the user's privileges or the state of an object or business process, you must create an ActionDelegate/URLActionDelegate pair (as described in the section on template processing).
2. The user clicks the link.
3. The URL arrives at the Windchill server and the action is performed. The invokeAction method in URLProcessor is used to process the request.
  - The ActionDelegate is used to check permission/availability of the desired action.
  - A FormTaskDelegate is used to perform the action.
4. A response page is generated and sent back to the browser by finding an TemplateProcessor and processing an HTML template. Either a success or failure page can be generated.

### Overview of Generating and Processing a Form

The following are the general steps that take place when a form is generated or processed:

1. A link is generated and displayed to the user. (Clicking the link results in the return of an HTML page containing a form.)
2. The user clicks the link leading to the HTML page containing the form.

3. An HTML page containing a form is generated. The generateForm method in URLProcessor is used to generate the form:
  - An ActionDelegate subclass is used to check permission and availability of the desired action (as described in the section on template processing.)
  - The HTML page containing the form is generated by finding a TemplateProcessor and processing an HTML Template.
4. The user fills out the form and submits the HTML form by clicking the appropriate button in the HTML page.
5. The URL and the HTTP POST generated by the form in the page arrive at the Windchill server, the data is processed, and the action performed. The processForm method in URLProcessor is used to process the URL and the form data:
  - An ActionDelegate subclass is used to check permission/availability of the desired action (as described in the section on template processing).
  - A FormTaskDelegate subclass is used to process the form data and perform the desired action.
6. A response page is generated and sent back to the browser by finding a TemplateProcessor and processing an HTML template. Either a success or failure page can be generated.

## HTTPState and Form Processing

This section discusses HTTPState and form processing specifically. For more general information on the HTTPState class and template processing, see the section on template processing.

During the processing of an HTML form that has been sent to the server through an HTTP POST, a FormActionDelegate is retrieved by the FormTaskDelegateFactory to process the data and perform any related actions. The FormActionDelegate is passed the HTTPState object that was instantiated by the URLProcessor. This HTTPState object should be used by the FormActionDelegate to keep track of the state of template processing. The FormActionDelegate aggregates the HTTPState object as a proxy so that calls like getContextObj() actually go to the getContextObj() of the HTTPState object to get the current object in the contextObj field.

One responsibility of the FormActionDelegate is to keep the context current by updating the HTTPState object. For example, if you were creating a business object, you would not have a context object initially. The object does not yet exist. However, after successfully creating the business object, you would like to update the context object field in the HTTPState object with the following call:

```
setContextObj( newObject );
```

The reason for updating the context, and the HTTPState object in particular, is that after the FormActionDelegate is finished and control is returned to the calling method in the URLProcessor class, the entries in the HTTPState object are used to determine what is done next. Specifically, the response page that will be generated is selected based on the values for the context in the HTTPState object. The template processor is selected based on these values. Then the HTTPState object is passed into the template processor to set the context in the template processor.

Following is a summary of the state of an HTTPState object during the processing of a form:

- URLProcessor instantiates a new HTTPState object and initializes it with the information from the query string and form data from the HTTP POST.
- The HTTPState object is used by the FormActionDelegateFactory to select the desired FormActionDelegate.
- The HTTPState object is passed off the selected FormActionDelegate:
  - The FormActionDelegate uses the information in the HTTPState object to perform the desired action.
  - The FormActionDelegate updates the HTTPState object before returning control to the URLProcessor.
- URLProcessor uses the current information in the HTTPState object to select the desired TemplateProcessor subclass from the TemplateProcessorFactory.
- The HTTPState object is passed into the TemplateProcessor subclass that is instantiated. The HTTPState object is passed in the bizData field on the wt.httpgw.HTTPRequest object.

## The invokeAction and processForm Methods

A question might arise as to why the invokeAction and processForm methods are used rather than URL TemplateAction. The big difference between invoking an action or processing a form and simply viewing information is that you need to check that your action is still valid and that you have permission to perform the action. As opposed to an interactive client, like the Java client, links can go stale in the sense that the action is no longer valid, but the page might not have refreshed. The invokeAction and processForm methods check these permissions and states; URL TemplateAction does not. Further, with invokeAction and processForm, there is a separation of task logic -- FormTaskDelegate class -- and presentation logic -- the TemplateProcessor subclasses. The URLTemplateAction method is focused on presentation.

## How to Invoke an Action

The following section gives brief explanations of how to perform specific actions in specific circumstances.

### Generating a Link to Invoke an Action

This link can be generated in two ways:

- If the link is part of the `createActionsBar` section, see the section on adding links to the navigation bar in the section on template processing.
- If the link is a stand-alone link, the following Windchill script API call will present the desired link:

```
<SCRIPT LANGUAGE=Windchill>
<!--
getUrlProcessorLink method=invokeAction action=<Desired Action>
  <Other Name/Value pairs>
-->
</SCRIPT>
```

The only caveat with this approach is that no permission-checking is done in the generation of this URL. This is acceptable if you are building a link to an action where there is no state or permission to check. Otherwise, you must be careful using this direct approach.

### Creating an ActionDelegate to Check Permissions and State

For a more detailed discussion of ActionDelegates, see the section on template processing.

If permission or state checking is required, create an ActionDelegate and use it to check permissions or validate the state of an object. This process is described in the section on Availability of the Page Depends on State or Permissions, earlier in this chapter.

If there are no permissions or state to check before allowing the processing of the action, do nothing. In the case that the ActionDelegateFactory cannot find an ActionDelegate based in the current context, then a very specific exception is thrown and grounded. The only possible problem is if your context actually does find an ActionDelegate when you did not intend to. Check that your action/object pair does not match an entry in service properties for an ActionDelegate if you do not want one found.

## Creating and Accessing a FormTaskDelegate to Perform an Action

This class can be created by editing a file directly or using the Rational Rose modeling tool:

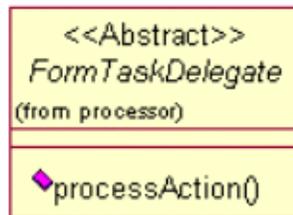
- If you create a subclass of FormTaskDelegate by editing a file, your subclass code should have the following lines:

```
import wt.templateutil.processor.FormTaskDelegate;  
  
public class NavigateFoldersTaskDelegate extends  
    FormTaskDelegate {
```

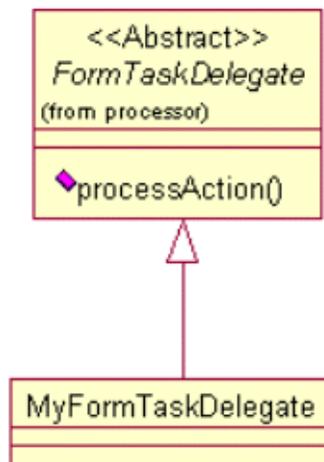
The subclass must have the following method:

```
public void processAction( ContentHTTPStream contentStream )  
    throws Exception {
```

- If you generate a subclass of FormTaskDelegate using Rational Rose, use the following steps:
  - a. Open WTdesigner.mdl in the/codebase/src/wt directory.
  - b. Open a class diagram in the package of your selection.
  - c. Add a new class and name it FormTaskDelegate:



- d. Add a new class and give it the desired name, for example, `MyFormTaskDelegate`, and have it subclass `FormTaskDelegate`:



- e. Generate the code using the menu selections **Tools > Windchill > System Generation**.
- f. Implement the abstract method `processAction`. The signature of `processAction` is as follows:

```
public void processAction (ContentHTTPStream contentStream)
    throws Exception
```

The `ContentHTTPContentStream` parameter is used to load any files that may have been sent in the HTTP POST. In addition, the calling method in `URLProcessor` sets the `HTTPState` object. This gives `MyFormTaskDelegate` the current values of the following:

- Action by calling `getContextAction()`
- Object (if any) by calling `getContextObj()`
- Classname (if any) by calling `getContextClassName()`
- The `queryString` using `getQueryData()`
- The form data using the `getFormData()`

When you have created the subclass of `FormTaskDelegate`, you must add an application context service property so the `FormTaskDelegateFactory` can find the new subclass. The typical entry looks like the following:

```
wt.services/svc/default/wt.templateutil.processor.
  FormTaskDelegate/<Action>/<Context Object or Context Class>/
  0=<Desired subclass of FormTaskDelegate>/duplicate
```

See the section, `Properties and Property Files` for information on how to add new properties to the system.

The role of the `FormTaskDelegate` is to perform some desired action based on the information in the `queryString` and the Form data. After processing the data and performing an action, the `FormTaskDelegate` returns control to the method in `URLProcessor` that called it. Prior to returning, the `FormTaskDelegate` should ensure that the current context stored in the `HTTPState` object is current. This is necessary because `URLProcessor` uses the `HTTPState` object to decide how to generate a response page. The following is a summary of the responsibilities of the `FormTaskDelegate`:

- Validate the Form Data/`queryString` and respond accordingly.
- Perform the desired action.
- Add any message to be passed back and displayed in the response page with the service methods.
- `addToResponseHeaders`, `addToResponseFooters`, and `addToResponseExceptions`. (Use `addToResponseExceptions` to capture non-terminal exceptions and display them in the response page.)

Corresponding Windchill script helper methods can be used to display the accumulated messages. These Windchill script calls have the following format:

```
showResponse<type of message>
```

An example is `showResponseHeaders`.

To format the presentation of these messages, override the following method in your `TemplateProcessor` subclass:

```
ShowResponseMessage(String message, Properties parameters,  
    Locale locale, OutputStream os)
```

- Update the context stored in the `HTTPState` so that `URLProcessor` can generate the correct response page based on the values in the `contextAction/contextObj/contextClassName` triad.

## Generating a Response Page

`URLProcessor` uses the `TemplateProcessorFactory` to locate a `TemplateProcessor` subclass and start processing an HTML template based on the values that you set in the `FormTaskDelegate` using the following calls:

- `setContextAction()`
- `setContextObj()`
- `setContextClassName()`

Be sure that all of the possible combinations of context in the `HTTPState` object are covered by entries in `service.properties`. Otherwise, when `URLProcessor` tries to generate a response page, an exception might be thrown. For the details, see the section on template processing.

## How to Generate and Process a Form

The following section gives brief explanations of how to generate and process forms in specific circumstances.

### Generating a Link to Return an HTML Page with a Form

This process is the same as that described earlier in this section for invoking an action.

### Creating an `ActionDelegate` to Check Permissions/State

This process is the same as that described earlier in this section for invoking an action.

## Generating an HTML Page Containing a Form

The process of generating an HTML page containing a form is slightly different than that of generating a standard HTML page that presents just information. Specifically, the page must contain the HTML FORM tags. The general steps to create this page are as follows:

1. Generate HTML FORM elements.
2. Add hidden FORM elements that might include context information.
3. Generate the URL to which you will submit the HTML form. Such a URL has the following format:

```
<FORM name="myForm" action=http://<Your Server Name>/  
  <Windchill CGI/Servlet>/wt.enterprise.URLProcessor/  
  processForm?action=MyAction method=POST>
```

To generate such a URL dynamically instead of hard coding it, you can use the following Windchill script call:

```
<FORM name="myform" action="<SCRIPT  
  language=Windchill>getURLProcessorLink method=processForm  
  action=MyAction</SCRIPT>" method=POST>
```

4. Generate the input fields in the HTML FORM. Following is a sample Windchill script call for generating input fields for the HTML FORM:

```
<INPUT name="number" type="text" value="<SCRIPT  
  language=Windchill>contextualValue  
  propertyName=number</SCRIPT>" >
```

The result of this call after processing by the Windchill system is as follows, assuming the context object of the page has an attribute named "number" and the value of that attribute is 11053:

```
<INPUT name="number" type="text" value="11053" >
```

Besides presenting the current value of the attribute, `contextualValue` also provides an input field (also called a *sticky* field) that retains the value entered, even if processing of the form fails when it is submitted. For example, assume a user enters a new value of 20000 for the "number" attribute in the HTML form, then submits it. If processing of the HTML FORM fails and the user is presented with the same form, the value of 20000, which the user just entered, is presented, not the original value of 11053.

The `getURLProcessorLink` is in `wt.enterprise.BasicTemplateProcessor`. The `contextualValue` is in `GenerateFormProcessor`. There are several service methods for generating HTML forms that reside in `GenerateFormProcessor`. Some of those methods are as follows:

- `EnumeratedTypeMenu propertyName=<attribute name>`, which provides a Select element for enumerated types.
- `listLifeCycles`, which provides a Select element for life cycle.

The other important fact is that `GenerateFormProcessor` subclasses `wt.templateutil.processor.DefaultTemplateProcessor`. This means that `GenerateFormProcessor` uses the same steps to process the HTML template. This has the following implications:

- The `DefaultHTML TemplateFactory` is used to locate the `HTMLTemplate`, including placing the entries to find your HTML template in the `htmltemplate.properties` file under the codebase directory.
- Overriding the `readContext` method to perform additional processing of the data returned to the server.
- Access to the `HTTPState` object and its service methods.

For further details, refer to the section on template processing.

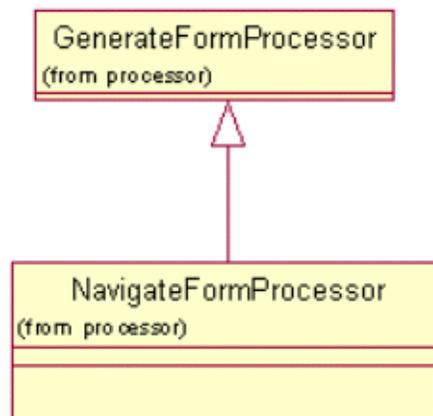
You can subclass `wt.templateutil.processor.GenerateFormProcessor` in either of the following ways:

- Code the file directly. In this case, your class must begin with the following lines:

```
import wt.templateutil.processor.GenerateFormProcessor
public class MyFormGenerator extends GenerateFormProcessor
```

- Model your subclass in Rational Rose using the following steps:
  - a. Open `WtdesignerTest.mdl` in the `/codebase/src/wt` directory.
  - b. Open a class diagram in the package of your selection.
  - c. Add `wt.templateutil.processor.GenerateFormProcessor` to your diagram.
  - d. Add a new class to the diagram and give it the name of your choice.
  - e. Have your new class subclass `GenerateFormProcessor`.

The following figure shows an example of the finished result:



## Submitting the HTML Form

An HTML form can be submitted in two ways. Both involve clicking a button; the difference is in how the button is interpreted.

- In one case, clicking the button submits the HTML form directly through an HTTP POST back to the server, as described here.

The standard manner to submit an HTML form is to add the following element:

```
<INPUT Type="submit" name="<Title on button>" >
```

This type of element results in a button with the title of your choice. When you click the button, the HTML form is read by the browser and sent to the URL that appears in the action attribute of the FORM tag.

- The other case involves using JavaScript to perform some intermediate steps before submitting the HTML FORM. This second, and more flexible way to submit the form, is to have an HTML tag for the button that looks like the following:

```
<INPUT type=button VALUE=" Ok " onclick="checkForm(this)">
```

In this case, a button is presented but, instead of the browser submitting the form directly, a JavaScript method is called. The JavaScript should look like the following:

```
<SCRIPT LANGUAGE="JavaScript1.1">
<!--
function checkForm( button )

...

}

//end hiding -->
</SCRIPT>
```

With a call like this, you can perform client-side validation of the entries.

## Creating a FormTaskDelegate to Perform the Action

This process is the same as that described earlier in this section for invoking an action.

## Generating a Response Page

This process is the same as that described earlier in this section for invoking an action.

## Properties and Property Files

As you have read in the previous sections, template processing relies heavily on the use of properties contained in properties files to locate the template, processor and other resources for an HTML client.

In general, properties for Windchill PDM out-of-the-box clients are grouped as follows:

**codebase/service.properties** – properties specifying the TemplateProcessor, ActionDelegate, URLActionDelegate, and FormTaskDelegate to use for a given action/object type

**codebase/htmltemplate.properties** – properties specifying the HTML template to use for a given action/object type

**codebase/htmlcomponent.properties** – properties specifying the HTMLComponent to be used for a given element

**codebase/wt.properties** – miscellaneous template processing properties such as colors of various page components (wt.html.color.\*), the text encoding used for various locales (wt.template.encoding.\*), and the default parameters used for various page tags (wt.templateutil.component.\*)

The first three of these files contain properties for what are typically known as “application context services.” These require special loading at runtime and are described in more detail in the following section.

The properties in wt.properties are simple name/value pairs that are loaded into a Java Properties class instance at runtime. You should not add to or modify this file directly because it will be hard to maintain your customizations if Windchill is reinstalled or upgraded. Instead, modify this file using your <WT\_HOME>/site.xconf file. For example, if you would like to change the maximum number of soft attributes displayed on the details page of parts from 6 to 10, you would need to modify the following property in wt.properties:

```
part.attributeNumber=6
```

To do this, you would perform the following steps:

1. Edit the file <WT\_HOME>/site.xconf and add the following line within the <CONFIGURATION> tag of the file.

```
<Property name="part.attributeNumber" overridable="true"
targetFile="codebase/wt.properties" value="10"/>
```

2. Regenerate the file <WT\_HOME>/codebase/wt.properties by executing the following command:

```
<WT_HOME>/bin/xconfmanager -p
```

After executing this command your wt.properties file should contain the following property:

```
part.attributeNumber=10
```

## Application Context Service/Resource Properties

These are properties, generally used by a factory class, for locating a delegate, service, or resource. They have one of the following formats:

```
wt.services/svc/default/<Service Name>/<Requestor>/<Selector> |  
null/<Search Order Number>=<Service Implementation  
Name>/<duplicate or singleton>
```

or

```
wt.services/rsc/default/<Resource Name>/<Requestor>/<Selector>  
| null/<Search Order Number>=<Resource Name>
```

The first format is used to locate a Java service or delegate class to perform a function or provide a service. The second format is used to locate a resource file -- for example, a HTML template file or an icon image file

### Definitions:

Service Type = the type of service or delegate referenced by this property

Resource Type = the type of resource referenced by this property

Selector = an identifier used to specify the context in which this service or resource is to be used (for example, an action name)

Requestor = the object class for which the service, delegate, or resource is to be used

Service Priority Number = a priority rating used to choose between valid delegates (see below)

Service Class Name = name of delegate or service class for the given Service Type, Selector, and Requestor

Resource Name = name of resource for given Resource Type, Selector, and Requestor

Duplicate or singleton = a flag to indicate whether the server should instantiate a shared instance of a delegate class or create a new one for each use. If neither is specified, duplicate will be used.

This is an example property for a template processor:

```
wt.services/svc/default/wt.enterprise.TemplateProcessor/AddAlternates/  
wt.part.WTPartMaster/0=wt.part.AlternatesLocalSearchProcessor/duplicate
```

where

```
Service Name = "wt.enterprise.TemplateProcessor"  
Requestor = the action name "AddAlternates"  
Selector = "wt.part.WTPartMaster"
```

**Note:** Any service class that incorporates an HTTPState object should be made duplicate. This would include instances of BasicTemplateProcessor, FormTaskDelegate, NavBarActionDelegate, and NavBarURLActionDelegate.

If a factory receives a request for a service or resource class for a given requestor object class but no property entry for that requestor class is found, the factory will attempt to find an entry for the parent class or interface of the requestor class. If no entry for the parent class or interface is found, a search will be made for an entry for the parent of the parent or interface, and so on. It could happen that entries for two or more parent classes or interfaces are found. If the entries have different service priority numbers, the one with the lowest number will be selected. If the entries have the same service priority number, the one selected is arbitrary.

To be loaded correctly at runtime, files containing application context service properties must be listed for one of the following properties in wt.properties:

```
wt.services.applicationcontext.WTServiceProviderFromProperties.  
defaultPropertyFiles  
  
wt.services.applicationcontext.WTServiceProviderFromProperties.  
customPropertyFiles
```

Property files will be loaded in the order listed, and files listed for defaultPropertyFiles will be loaded before those for customPropertyFiles. If the same property is found in more than one file, the value for the one loaded last will be used. Any custom properties should be placed in the latter list.

Except for the need to separate application context service properties from ordinary properties and the effect of load order, the grouping of properties into various properties files is unimportant to the system and based primarily on ease of maintenance. If a TemplateProcessor property is put in the htmltemplate.properties file instead of service.properties the system will still find it.

Many of the more heavily customized service property files are not created or edited directly but instead are generated from xml files. XML files used to generate property files have the same name as the associated property file but have the additional extension ".xconf". For example, the XML file used to generate service.properties is called service.properties.xconf. See the the Application Developer's Guide for more information on xconf files.

If you need to add service property entries for your custom HTML clients, we recommend you put them in a new properties file or files used only for your

customizations. This file should be added to the list of files for WtServiceProviderFromProperties.customPropertyFiles.

There are a number of ways your custom service properties can be managed such that they are not overwritten if Windchill is upgraded or reinstalled, as described in the Windchill Application Developer's Guide.

Probably the simplest way to add your own properties file to the list for customPropertyFiles is to add an entry to your <WT\_HOME>/site.xconf file. For example, assume you would like to create your own properties file named <WT\_HOME>/codebase/com/MyCompany/MyCompany.properties. You can do so by adding an entry similar to the following to site.xconf:

```
<Property targetFile="codebase/wt.properties"
           value="htmltemplate.properties, ...,
com/ptc/windchill/pdmlink/tutorials/pdmlinkTutorials.properties,co
m/MyCompany/MyCompany.properties"
name="wt.services.applicationcontext.WtServiceProviderFromProperti
es.customPropertyFiles" />
```

**Note:** The text in italics in this example should be replaced with the existing list of customPropertyFiles in your wt.properties file. The objective is to add your new file to the existing comma-separated list of files for that property.

After adding this entry, back up your wt.properties file and rebuild your system property files by running the command:

```
xconfmanager -p
```

This command will add the file <WT\_HOME>/codebase/com/MyCompany/MyCompany.properties to the comma-separated list of files for the property wt.services.applicationcontext.WtServiceProviderFromProperties.customPropertyFiles in

<WT\_HOME>/codebase/wt.properties.

**Note:** Tags you add to site.xconf should be placed within the existing <Configuration> tag.

Once you have added your property file to the customPropertyFiles list, you can create and add entries to your properties file directly, or you can create and add properties to it using the site.xconf file. For example, if you wanted to add a new

template template, template processor, ActionDelegate, and URLActionDelegate for the action “MyAction” you could do so by adding tags similar to the following to site.xconf:

```
<Resource context="default"
name="wt.templateutil.DefaultHTMLTemplate"
    serviceProvider="wtCustom"

targetFile="codebase/com/MyCompany/MyCompany.properties">
    <Option cardinality="duplicate"
        requestor="java.lang.Object" selector="MyAction"
        resource="templates.MyCompany.MyTemplate" />
</Resource>

    <Service context="default"
name="wt.enterprise.TemplateProcessor"
    serviceProvider="wtCustom"
        targetFile="codebase/com/MyCompany/MyCompany.properties"
    >
        <Option cardinality="duplicate"
            requestor="java.lang.Object" selector="MyAction"
            serviceClass="com.MyCompany.MyTemplateProcessor" />
        </Service>

    <Service context="default" name="wt.enterprise.ActionDelegate"
        serviceProvider="wtCustom"
        targetFile="codebase/com/MyCompany/MyCompany.properties"
    >
        <Option cardinality="duplicate"
            requestor="java.lang.Object" selector="MYACTION"
            serviceClass="com.MyCompany.MyNavBarActionDelegate" />
        </Service>

    <Service context="default"
name="wt.enterprise.URLActionDelegate"
    serviceProvider="wtCustom"
```

```

        targetFile="codebase/com/MyCompany/MyCompany.properties"
    >
        <Option cardinality="duplicate"
            requestor="java.lang.Object" selector="MYACTION"

serviceClass="com.MyCompany.MyNavBarURLActionDelegate" />
    </Service>

```

These four entries will cause the system to add four properties to the file `<WT_HOME>/codebase/com/MyCompany/MyCompany.properties` as follows:

```

wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/MyAction/
    java.lang.Object/0=MyCompany.MyTemplate
wt.services/svc/default/wt.enterprise.TemplateProcessor/MyAction/
    java.lang.Object/0=com.MyCompany.MyTemplateProcessor/duplicate
wt.services/svc/default/wt.enterprise.ActionDelegate/MYACTION/
    java.lang.Object/
    0=com.MyCompany.MyNavBarActionDelegate/duplicate
wt.services/svc/default/wt.enterprise.URLActionDelegate/MYACTION /
    java.lang.Object/
    0=com.MyCompany.MyNavBarURLActionDelegate/duplicate

```

If the property file does not exist, the system will create it.

Note that the resource and service class paths should be given relative to `<WT_HOME>/codebase`.

For these changes you make to `site.xconf` to take effect, you must again rebuild your property files using the command:

```
<WT_HOME>/bin/xconfmanager -p
```

We recommend you back up any property files you are modifying before running this command.

More than one `<Option>` tag may be included within a `<Service>` or `<Resource>` tag. The “selector” attribute values for `ActionDelegates` and `URLActionDelegates` should be all upper case.

See the file `<WT_HOME>/codebase/service.properties.xconf` for additional examples.

## Application Context Service Properties for Soft Types

If you need to add typed service property entries for your custom HTML clients, we recommend you put them in a new properties file or files used only for your customizations. This file should be added to the list of files for `TypeBasedServiceProviderFromProperties.defaultPropertyFiles`. For example, if you would like to create a custom typed property file named `codebase/com/MyCompany/MyCompanyTypedServices.properties` you would follow the steps below, which are very similar to those for creating a non-typed application context property file.

1. Create a file called `<WT_HOME>/codebase/com/MyCompany/MyCompanyTypedServices.properties.xconf`. Insert the following text into the file:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Configuration
  SYSTEM "xconf.dtd">
<Configuration targetFile="codebase/wt.properties">
<!--
Ensure that the file
com/MyCompany/MyCompanyTypedServices.properties is appended to the
list of typed services property files-->
<Property
default="com/MyCompany/MyCompanyTypedServices.properties" name="
wt.services.applicationcontext.TypeBasedServiceProviderFromPropert
ies.defaultPropertyFiles "/>
</Configuration>
```

2. Make your new xconf file known to the system by executing the following command:

```
<WT_HOME>/bin/xconfmanager-i
codebase/com/MyCompany/MyCompanyTypedServices.properties.xconf
```

This will add your xconf file to the list of xconf files in `<WT_HOME>/declarations.xconf`

3. Execute the directives in your new xconf file by executing the command:

```
<WT_HOME>/bin/xconfmanager -p
```

This will add the property file  
<WT\_HOME>/codebase/com/MyCompany/MyCompanyTypedServices.properties to the list of typed service property files in  
<WT\_HOME>/codebase/wt.properties as follows:

```
wt.services.applicationcontext.TypeBasedServiceProviderFromProperties.default
PropertyFiles =
com/ptc/core/command/server/delegate/ServerCommandDelegate.properties,com
/ptc/core/meta/type/command/TypeBasedCommandDelegate.properties,. . .
,typedservices.properties,
com/MyCompany/MyCompanyTypedServices.properties
```

**Note:** Steps 2 and 3 can be combined as follows:

```
<WT_HOME>/bin/xconfmanager -i
codebase/com/MyCompany/MyCompanyTypedServices.properties.xconf -p
```

4. Create your custom properties. Assume you would like MyCompanyTypedServices.properties to contain the following property mapping the dropdown action list on the Details pages for the soft type SpecificationDocument to the tree SPECIFICATION\_DOCUMENT\_DROPDOWN\_ACTIONS in the file NavigationAndActions.xml:

```
wt.services/rsc/default/wt.templateutil.objectPropsActionDropDown/DROPDO
WN/WCTYPE|wt.doc.WTDocument|SpecificationDocument/
0=SPECIFICATION_DOCUMENT_DROPDOWN_ACTIONS
```

This property can be created in one of two ways:

- a. You can create the file  
<WT\_HOME>/codebase/com/MyCompany/MyCompanyTypedServices.properties and add the property to it by hand.
- b. You can add entries to the file MyCompanyTypedServices.properties.xconf such that the system will create the properties file and add the properties to it. To do this you would add the following tag to the xconf file:

```
<Resource context="default"
  name="wt.templateutil.objectPropsActionDropDown"
  serviceProvider="typeBased"
  targetFile=
  "codebase/com/MyCompany/MyCompanyTypedServices.properties">
  <Option
    requestor=
      "WCTYPE|wt.doc.WTDocument|SpecificationDocument"
      resource="SPECIFICATION_DOCUMENT_DROPDOWN_ACTIONS"
      selector="DROPDOWN" />
</Resource>
```

After adding this entry, run the following command to create your properties file and append this property to it:

```
<WT_HOME>/bin/xconfmanager -p
```

We recommend you back up any property files you are modifying before running this command.

**Note:** Any tags you add to your .xconf file should be placed within the existing <Configuration> tag and that more than one <Option> tag may be included within a <Service> or <Resource> tag.

## HTML Client Services

Many HTML client services are available. Several of them are described in more detail in other sections of this chapter to put them in context and show them in examples. The following services are described in this chapter:

- DefaultTemplateProcess, which is described in the section on template processing
- GenerateFormProcessor, which is described in the section on template processing
- HTMLTemplateFactory, which is described in the section on HTML template factories
- SubTemplateProcessing, which is described in this section
- ProcessorService, which is described in this section
- HTML components, which are described in the section on components later in this chapter
- HTML tables, which are described in the section on tables later in this chapter
- HTML table service, which is described in the section on tables later in this chapter
- HTML help page service, which is described in the section on adding an HTML help link later in this chapter

### SubTemplateProcessing

With the formal encapsulation of the state of a template processor (in the HTTPState object) comes the ability to pass off the template processing to a subtemplate processor. The main purpose of this service is to allow the componentization of the HTML templates and consequently the template processors themselves. This provides code reuse not through inheritance but through delegation. Thus, when several HTML pages share a common section, this section can be factored out into a *subtemplate* and the Windchill script calls in that subtemplate can be factored out into a subtemplate. This provides better code reuse, fewer bloated classes near the top of the inheritance hierarchy (such as BasicTempateProcessor), and easier HTML template maintenance.

There are two ways to using subtemplate processing:

- Calling the subtemplate service directly in the Java code
- Calling the subtemplate service from a Windchill script call

To call the subtemplate service directly in Java code, you would use code similar to the following:

```
Properties parameters = new Properties();
parameters.put( "action" , "<New ContextAction>" );
processSubTemplate( parameters, locale, os );
```

To make the call to the subtemplate service through a Windchill script, you would make the following call:

```
processSubTemplate action=<New ContextAction>
```

The results of both calls is the same. A template processor is retrieved and an HTML template is retrieved. The template processor and the HTML template are retrieved using factories that use the current context as represented in the HTTPState object. The current context object or context class name is used with that action that was just passed in.

For example, if the Properties page for a WTPart is being generated, the context object is the WTPart being presented on the page and the current contextAction is "ObjProps".

Assume the following Windchill script method was encountered in the HTML template:

```
processSubTemplate action=showSubParts
```

Then a template processor specified by the property would be looked for:

```
wt.services/svc/default/wt.enterprise.TemplateProcessor/
showSubParts/
wt.part.WTPart/0=<Sub-TemplateProcess>/duplicate
```

The HTML template that would be looked for would be specified by the following property:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
showSubParts
wt.part.WTPart/0=<Path to HTML template from codebase>
```

The HTML template returned would be processed by the template processor that was returned and the Windchill script call would be replaced with the output of the processing subTemplate.

After the subTemplate service is finished, the contextAction is returned to the original action, in this case, "ObjProps".

If you want to change the context object or the context class name before calling the subTemplateService, you perform that in code you write; this is not supported in the Windchill script call. In such a case, you might want have the following Windchill script method in your template processor :

```
public void processMySubTemplate( Properties parameters,
    Locale locale, OutputStream os )
{
    // Some code that selects the new contextObj, newContextObj.
    ...
    //
    Object origContextObj = getContextObj();
    setContextObj( newContextObj)
    processSubTemplate( parameters, locale, os );
    setContextObj(origContextObj );
}
```

Then you would call the following specialization of the subTemplate service through the Windchill script call:

```
processMySubTemplate action=<New ContextAction>
```

## ProcessorService

This service is very similar to the subTemplateProcessing service. The main difference is that no subtemplates are processed. The Windchill script call results in a method on a "service" to be invoked. The invoked method in the service behaves as if it were a method in the TemplateProcessor.

With the ProcessorService, a method can be taken out of a template processor for general use, or removed from a common parent class. This allows the TemplateProcessor class to be more cohesive and allow organizing common services that all TemplateProcessors use. This creates a more componentized, cohesive, object-oriented code with greater code reuse.

An example of an implementation of the ProcessorService is the HTML table service (described in more detail later in this chapter). It is a context-based service which exists outside of any template processor, is very componentized, and available to all template processors without burdening BasicTemplateProcessor.

## Using the ProcessorService

To use the ProcessorService, find a service that you could factor out into a general service to be used by several template processors. Then model a subclass of the wt.templateutil.processor.ProcessorService class and generate the code.

**Note:** You must model this subclass; otherwise the code will be incomplete.

Assume in this case that the subclass is called `com.myCompany.CustomProcessorService`. The methods that the service is presenting must implement the standard Windchill Script signature as follows:

```
public void <ServiceMethod Name>( Properties parameters,
    Locale locale, OutputStream os )
```

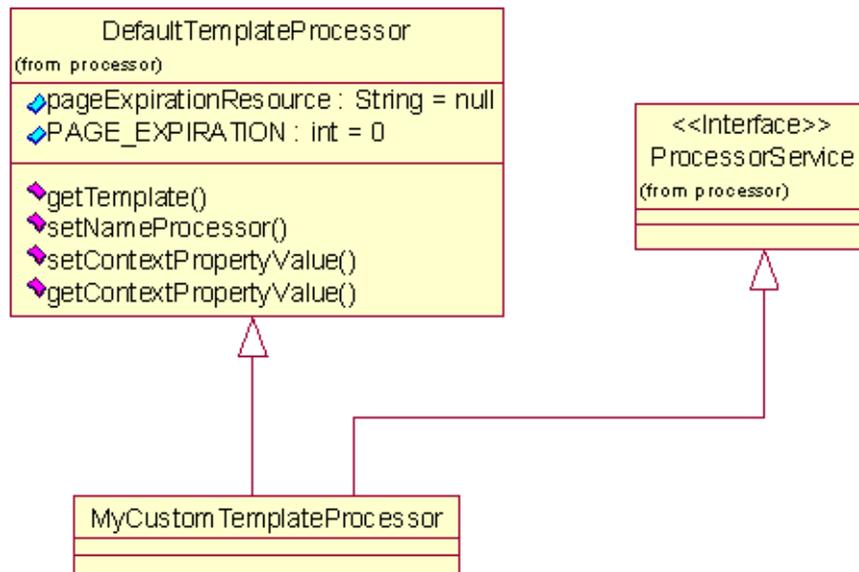
To access your new Service method, make the following Windchill script call:

```
useProcessorService service=com.MyCompany.CustomProcessorService
method=<ServiceMethod Name>
```

## Making a TemplateProcessor into a ProcessorService

You may have a template processor that has several methods in it that could be of use to other template processors. In that case, you could factor out the relevant methods into a ProcessorService. If there are reasons you should not factor out the method, an alternative is to make the template processor into a ProcessorService. To enable a template processor to be a processor service, the template processor must be a subclass of BasicTemplateProcessor. There are two ways (described next) to enable your template processor, depending on whether it is modeled in Rational Rose. Both are simple to implement because the methods in the ProcessorService interface are already implemented by BasicTemplateProcessor. The only additional requirement is that your TemplateProcessor is tagged as a ProcessorService.

- If your TemplateProcessor is modeled in Rose, you simply need to have your modeled TemplateProcessor implement the ProcessorService interface in the model. Then regenerate your code. None of the methods in the TemplateProcessor should be affected by the regeneration other than some of the methods of the Externalizable interface. Now your TemplateProcessor should be able to be used as a ProcessorService.



- If your TemplateProcessor is not modeled in Rose, add the implements wt.templateutil.processor.ProcessorService to the class definition line. For example, the following code:

```
public class MyTemplateProcessor extends  
    DefaultTemplateProcessor
```

should be changed to the following:

```
import wt.templateutil.processor.ProcessorService;  
  
public class MyTemplateProcessor extends  
    DefaultTemplateProcessor  
    implements ProcessorService
```

Now your TemplateProcessor is ready to be used as a ProcessorService.

## Adding an HTML Help Link

You can add links to help on HTML pages in several ways:

- Through hyperlinked text in the HTML page
- Through a button containing localized text on the HTML page
- Through the global navigation bar

This section describes how to add each kind of help link.

### Adding Help Using Hyperlinked Text

#### The Windchill Script API

To add hyperlinked, localized text that points to a help page, the most basic Windchill script call to make is `addHTMLHelpLink`.

Many options can be specified in this script call. The current list of options for configuring the help link, through name/value pairs in the Windchill script call, is as follows:

**HelpContext**

Replaces the context action that is used to locate the help page.

**PresentHelpLabel**

Set to true or false to turn on or off the presentation of a text label. Default is on.

**HelpLabelResource**

Identifies the resource bundle that is used to locate the localized text label.

**HelpLabel**

Identifies the resource key that is used to locate the localized text label.

**ADD\_HELP\_ICON**

Set to true or false option to turn on or off the presentation of an icon with the help link.

**HELP\_ICON\_SELECTOR**

Replaces the context action that is used to locate the icon from `service.properties`.

**HELP\_ICON\_POSITION**

Set to `LEFT` or `RIGHT` to position the icon to the left or right of the text label. Default positioning of the help icon is to the left of the text label.

The following are examples of addHTMLHelpLink script calls:

```
addHTMLHelpLink HelpContext=<help context name> HelpLabel=<key in
resource bundle>
  HelpLabelResource=<resource bundle>
```

```
addHTMLHelpLink HelpLabel=<key in resource bundle>
HelpLabelResource=<resource bundle>
```

The only difference between these two calls is the name/value pair HelpContext=<Help Context Name>, for example, HelpContext=ViewPart. Including this name/value pair overrides the default HelpContext of the page. More detail will be given about this in the section Adding a Property for the Help Page.

## Adding the Resource Bundle Entry

Based on the entries shown in the preceding section, you would need to update the resource bundle listed in the HelpLabelResource=<resource bundle> name/value pair. An entry that matches the string passed in the HelpLabel=<key in resource bundle> name/value pair must appear in the resource bundle.

For example, assume you used the following call:

```
addHTMLHelpLink HelpLabel=VIEW_PART HelpLabelResource=
  wt.enterprise.enterpriseResource
```

You would then need the following entries in the wt.enterprise.enterpriseResource\*.java resource bundles:

```
public final static String VIEW_PART=                "142";

{VIEW_PART                "Help for Viewing a Part"},
```

In the preceding entry, the value "142" is somewhat arbitrary. The only restriction on this value is that some other key in the same resource bundle is not using "142" as its String value. For most resource bundles, the setting of the key's String value is at the top of the file and you scroll down to the bottom of the list and choose the next, unused number.

## Adding a Property for the Help Page

### Selecting a Properties File for the Entry

For each different help page, you must add an application context service property. See the section, Properties and Property Files for information on how to add new properties to the system.

### The Syntax of the Entry

The following is the syntax to use when adding an entry to a properties file for an HTML help link:

```
wt.services/rsc/default/wt.templateutil.processor.HelpHTMLTemplate/  
<help context>/<context object of HTML page>/  
0=<relative path of help page file>#<anchor name>
```

#### **help context**

The default value of this entry is the context action of the HTML page, the value returned from the `getContextAction` method. This is the same value that is passed in either the query string or POST data on the required "action" parameter. To override this default, you can use the `HelpContext` parameter in the Windchill script API call. If you pass in a value in the Windchill script call, this value is used to locate the entry in the properties file.

#### **context object of HTML page**

This is the context object/class of the HTML page. The selection of this value follows the following process:

- a. If `getContextObj() != null`, the current context object is used.
- b. If `getContextClassName() != null`, this class is used to define the class of the context object that is used.
- c. If both of the preceding are null, `java.lang.Object` is used.

### relative path of help page file

The HTML file that is the help file must be located under the /Windchill/codebase/ directory. The relative path specified is assumed to begin under this directory.

Keep in mind that localization for the help pages is done off the directory and not the file. For example, you would see the following localization scheme for the help pages:

```
/wt/clients/prodmgmt/help_en/viewpart.htm  
/wt/clients/prodmgmt/help_de/viewpart.htm  
/wt/clients/prodmgmt/help_fr/viewpart.htm  
...
```

This differs from the pattern in which localization is done on the HTML template pages, that is, files are localized directly, as shown in the following examples:

```
/wt/clients/prodmgmt/<userinp>help</userinp>/viewpart.htm  
/wt/clients/prodmgmt/<userinp>help</userinp>/viewpart<userinp>_  
de</userinp>.htm  
/wt/clients/prodmgmt/<userinp>help</userinp>/viewpart<userinp>_  
fr</userinp>.htm  
&hellip;
```

Assume you have the following set of localized help files:

```
/wt/clients/prodmgmt/help_en/viewpart.htm  
/wt/clients/prodmgmt/help_de/viewpart.htm  
/wt/clients/prodmgmt/help_fr/viewpart.htm  
...
```

In this case, your entry in the properties files should have the following relative file path:

```
wt.clients.prodmgmt.help.viewpart
```

### anchor name

If your help page has anchors in it and you would like your help page to appear at that anchor, the HTML help service supports this. Assume you have the anchor VIEWPARTCONTENTS in the following file:

```
wt/clients/prodmgmt/help_en/viewpart.htm
```

In this case, you would modify your entry in the properties file as follows:

```
wt.clients.prodmgmt.help.viewpart#VIEWPARTCONTENTS
```

## Adding an Icon to the Help Link

As indicated earlier in this section by the parameters that could be passed in the Windchill script call to generate a link to a help page, an icon can be included in the link to the help page. If the following entry in `wt.properties` is set to `false`:

```
wt.htmlhelp.icon.enable=true/false
```

you will have to specify that you want to have an icon appear by passing in the name/value pair `ADD_HELP_ICON=true`.

To locate the image to present with the help link, the current context of the template processor is used in conjunction with the `ContextBasedLocalizedResourceSrv` (see Javadoc for more details) to find a localized version of the image. Otherwise, the non-localized version of the image is returned. `ContextBasedLocalizedResourceSrv` looks for the following entry in `service.properties` to find the correct image file:

```
wt.services/rsc/default/LocalizedResourceService/  
  <Help Context>/<Context Object Class>/0=  
  <Relative Path from Codebase>
```

### Help Context

The action context to use for the help icon. The default is "HELP". To override this default, pass in the name/value pair in the Windchill script call `HELP_ICON_SELECTOR=<IconAction Context>`.

### Context Object Class

The current context object of the page.

### Relative Path from Codebase

The path to the image relative to the codebase directory. The file separators in the path must be replaced with period (.). For example, the following relative path:

```
wt/clients/images/helphtml.gif
```

should be entered as the following:

```
wt.clients.images.helphtml.gif
```

As opposed to the entries for the HTML templates, you do include the file extension.

One final option with the addition of the an icon to the help link is whether you would like the icon on the right or the left of the text (if text exists). This can be controlled by passing in the name/value pair `HELP_ICON_POSITION=right/left` in the Windchill script call.

## Adding Help Using a Button

### The Windchill Script API

To add help that is launched through a button, you perform one more step than you did for adding the hyperlinked text. For this option, you must make two entries to the HTML template.

#### Adding the Button

To add a button with localized text to your HTML page, make the following call:

```
<INPUT type=button VALUE="<SCRIPT LANGUAGE=Windchill>
  <!--
  getResourceString textToken=HELP_BUTTON textResourceBundle=
    wt.federation.admin.adminResource
  -->
  </SCRIPT>" onclick="displayHelp()">
```

For this to work in Netscape, you must have the button inside the `<FORM>` and `</FORM>` tags. That is, the button must be inside an HTML form block.

#### Adding the JavaScript Function to Launch the Browser

The JavaScript to start the help page should look like the following:

```
function displayHelp( )
{
    var url = "<SCRIPT LANGUAGE="Windchill">getHTMLHelpURL
      HelpContext=<Help Context></SCRIPT>"
    open(url, "help_browser",
      "resizable=yes,scrollbars=yes,menubar=yes,toolbar=yes,
      location=yes,status=yes ");
}
```

Note that the `HelpContext` parameter is not required unless you would like to override the default context of the HTML page being generated.

Adding the resource bundle entry and the help page property are the same as described in the earlier section.

## Adding Help Using the Global Navigation Bar

The call to create the global navigation bar in the HTML client also results in the global navigation bar trying to add a help link for your page. The help link is created if the following conditions are true:

- The following entry is in wt.properties:

```
wt.globalNavigationBar.htmlhelp.icon.enable=true
```

If you set the value to false, the default is to not present the link.

- You have an entry in one of the properties files that corresponds to the context of the HTML page. That is, there is an entry matching the value of `getContextAction` and either `getContextObj` or `getContextClassName` (if `getContextObj() == null`).

### Adding the Link

You must add an entry in a properties file that matches the context of the HTML page. The details of adding this entry are described in detail in the preceding section, [Adding a Property for the Help Page](#). See that documentation for the specifics.

### Overriding the Defaults of the HTML Help Link

Several defaults are used in the link to the help that appears in the global navigation bar. The name/value pairs that can be passed in the Windchill script call to affect the presentation of the help page link are covered in detail in the preceding section on the Windchill script API. The parameter names and effects are the same.

## Overview of HTML Components, Table, and Table Service

This section gives a general overview of HTML user interface (UI) components, the HTMLTable, and the table service, and how they interact. This section is followed by individual sections on each, giving specific instructions for common actions. Generally, these four sections should be sufficient for you to gain a working knowledge of these areas and be able to make customizations to them. If, however, you want further details about how this functionality was actually implemented in Windchill, two sections at the end of this chapter describe the implementation of HTML components and the implementation of HTMLTables and table services.

To use these sections related to HTML components and tables, you are assumed to be familiar with the HTML client, in particular, the basics of HTML template processing and how the Windchill script calls are handled (as described earlier in this chapter).

It is also assumed you are familiar with HTML, HTML tags, attributes of the tags, and the structure of an HTML page and the nesting of HTML tags. It is helpful to be familiar with the Swing JTable and the associated classes, for example, TableColumnModel, TableColumn, and especially the TableModel. Some background on the Swing JTable, relevant to these sections, will be provided.

The HTML components, the HTMLTable, and the HTML table service all work together to generate the HTML to present Windchill information.

HTML components exist and are used outside of the HTMLTable and its subclasses. For example, HTML components are used in the objectPropertyValue and the objectPropertyName Windchill script methods. These methods present individual pieces of information on a Windchill business object. The HTML components are used to generate the HTML for the presentation. In fact, HTML components can be used anywhere HTML is being generated. The main purpose of HTML components is to provide a mechanism to create a UI component that produces HTML. Thus, if there is a situation where you are generating the same HTML in several places, the best approach might be to create an HTML component and use the HTML component to generate the HTML.

HTMLTable provides the ability to present tabular information in the HTML client. The HTML table service provides the ability to manipulate the HTMLTable presentation directly through the Windchill script calls. This tandem allows the HTML-based client to present tables through a standard process and allows those tables to be highly configurable without affecting Java code. In the generation of the table, the headers and cells in the table are generated using HTML components.

## HTML Components

HTML components represent individual UI components. There are essentially two types of HTML components: base components and custom components

The base components represent a specific HTML tag. The base components know what attributes the HTML tag has and also how to initialize the attributes with defaults from entries in `wt.properties`.

Customized components have some business/UI logic. They can be an extension of a base component. They can be the composite of several base HTML components. An example of a custom component is the `IconFirstColumnComponent`. This HTML component is the composite of several other HTML components. The two subcomponents of the `IconFirstColumnComponent` have the business/UI logic to find the correct icon for the object being presented by the component and also know how to make a link to the properties page of the business object being presented by the HTML component.

The `HTMLComponentFactory` finds the correct HTML component to use. This context-based factory uses entries in properties files, usually `htmlcomponent.properties`, to select the appropriate HTML component based on the object to present and a `String` that represents the context. The `objectPropertyName` and the `objectPropertyValue` Windchill script methods use the `HTMLComponentFactory` to find the correct HTML component. The `HTMLTable` also makes use of the `HTMLComponentFactory` if an `HTMLTableColumn` does not provide an HTML component to be used for the cells in that column of the table.

## The Components

The HTML components are styled after the Java UI AWT components. A parent class has most of the default behavior and defines the base interface for an HTML component. The class `wt.templateutil.components.HTMLComponent` plays the same role that `java.awt.Component` does. All of the HTML components subclass `HTMLComponent`. This mirrors the AWT where all of the UI components subclass the `java.awt.Component` class.

## The Primary API

A base API is defined by the `HTMLComponent`. This API is covered in the Javadoc for those classes. The primary API methods and their usage are briefly discussed here as follows:

```
public void init( String selector, Object value,  
HTMLComponentFactory componentFactory, String mode,  
Properties props )
```

This method is responsible for initializing the structure of the HTML component. For example, if this component has subcomponents, this method is responsible for initializing and adding the subcomponents.

**public String show( Object value, Properties formData, PrintWriter out, Locale locale )**

This method is responsible for generating the HTML to present in the browser. The return type, a String, should contain the HTML to display in the browser. The component should not print the String to output. The services that call the HTML component assume that the HTML will be returned to them for possible further manipulation. The only exception is HTMLTable. It has the option of printing to output directly or returning a String.

The show method calls the following three methods in succession to generate the three parts of an HTML element:

- startComponent, which generates the starting tag with attributes of the tag
- showSubComponents, which generates text between the starting and ending tag
- endComponent, which generates the ending tag

**public String startComponent( Object value, Properties formData, PrintWriter out, Locale locale )**

This method is responsible for generating the starting HTML tag and the attributes of the HTML tag. For example, in the following HTML:

```
<TD BGCOLOR=blue ALIGN=RIGHT>My Favorite Part</TD>
```

the startComponent() method is responsible for generating the <TD BGCOLOR=blue ALIGN=RIGHT> tag.

**public String showSubComponents( Object value, Properties formData, PrintWriter out, Locale locale )**

HTML components can contain other HTMLComponents, just as Java AWT components can contain other Java AWT components. For the HTML components, a vector contains all of the HTML subcomponents. When showSubComponents is called, this list is enumerated over and the show() method is invoked on all of the HTML subcomponents in the vector. In the preceding HTML String, the text My Favorite Part would be generated by the call to showSubComponents().

**public String endComponent( Object value, PrintWriter out, Locale locale )**

This method is responsible for printing the end tag of the HTML tag. In the preceding HTML String, the text that is put out by this method is </TD>.

**public void setTagListArray()**

This method must be overridden by an HTML component that is presenting a new HTML tag with tag attributes, that is, a new base component. The HTMLComponent class does not have its own tag. However, the class HTMLFont represents the base HTML class that presents the <FONT> </FONT> HTML element. Here the tag list represents the supported attributes of the HTML tag FONT. The class DefaultHTMLFont that subclasses HTMLFontComponent does not need to override this method. It inherits its HTML tag and attributes from the HTMLFont class.

### **public void setTagValue( String tagID, String tagValue )**

This method allows you to set values for the individual tag attribute values. For example, you might want to set the color of the FONT to be green. Then the following call:

```
htmlFontInstance.setTagValue( HTMLFont.COLOR,  
    BasicTemplateProcessor.getWCColor("f-hilite"));
```

Sets the value of the FONT attribute, COLOR, to the value of green.

Some default behavior also defined by the HTMLComponent class should be inherited by all of the subcomponents. Following are some of those default behaviors:

- The initialization of tag attributes by default entries in wt.properties
- The processing of the properties passed in the init method call
- The calling of subcomponents during the showSubComponents call

### **Reading Values for the Tag Attributes**

This section describes reading context-enabled, default values from wt.properties for the tag attribute values. One of the goals of the HTML components is that the components can share a set of default values across the HTML-based Windchill client where those default values are not in Java code. Further, the default values must be able to be context-based. That is, depending on the context, a different set of default values applies for the component.

In this case, the default values are in the wt.properties file. The format of these entries is as follows:

```
<Service Name>.<HTML Tag>.<Tag Attribute>=value
```

For example, the following entry specifies that the HTML table would have a border of size=0:

```
wt.templateutil.component.TABLE.BORDER=0
```

If desired, the setServiceName( String newServiceName ) method in the HTMLComponent can be used to override the default service name, wt.templateutil.component. This allows referencing a difference set of default values. Thus, based on the context you can have different default values.

The code that performs this initialization is in the init method of HTMLComponent. The following call, from within the init method of an HTML component that subclasses HTMLComponent, invokes this service:

```
super.init( selector, value, componentFactory, mode, props );
```

The HTML tag is retrieved from the subcomponent by defining the method public String getTag(). The list of tag attributes to be read are set by the public void setTagListArray() method.

## The Component Factory

### Basic Usage

The factory class, `wt.templateutil.components.HTMLComponentFactory`, is responsible for retrieving HTML components based on a String specifying some context, and a class specifying the class of the object to be presented. For example, if an attribute of type `java.sql.Timestamp` is to have its value displayed, then the following pair:

```
Context = null  
Class = java.sql.Timestamp
```

should find an HTML component that knows how to present an object that is of type `java.sql.Timestamp` in a desired HTML format. The factory looks for these values in the file `/codebase/htmlcomponent.properties`. The general entry in the file `/codebase/htmlcomponent.properties` looks like the following:

```
wt.services/rsc/default/wt.templateutil.components.HTMLComponent/  
<Context>/<Display Class>/0=<HTML Component class to be used>
```

The HTML component that is returned has not been initialized. It is simply instantiated.

### The HTMLValueObject/WTAttribute Classes

When an HTML component presents the value of an attribute of a business object, you might need more information than simply the String value of the attribute. If you were to return only the String value of an attribute, you could not perform many presentation activities. For example, if you wanted to make a link to the business object from the name attribute of the business object, you could not do so with only the value of the attribute. You would also need the business object itself.

To facilitate this need within the Model-View-Control (MVC) world of the HTMLTable, following the lead of the JTable, an interface has been created for wrapper classes that allows sending enough information to generate the view without the view directly interacting with the model. The HTMLComponentFactory can work with the HTMLValueObject/WTAttribute classes. If an instance of either the HTMLValueObject or the WTAttribute is passed in as the contextObj, the HTMLComponentFactory knows how to retrieve the true contextObj from the HTMLValueObject/WTAttribute through the `getDisplayObject` method.

The primary methods are as follows:

#### **public Object getDisplayObject()**

This method returns the object that an HTML component should present. This method is used by the HTMLComponentFactory to select the correct HTML component based on the class of the displayObject.

**public Class getDisplayClass()**

This method returns the class of the object that an HTML component should present.

**public Object getSource()**

In the case of an attribute of business class, this method returns the actual business object.

**public String getFormElementName()**

If the current object is being presented in an HTML form, this method retrieves the name to use in the HTML form for either the INPUT or the SELECT element.

## The HTMLTable

The role of the HTMLTable class is to print out an HTML table. The HTML table is based on the JTable in the Java Swing package. The main point is that the HTMLTable uses the Model-View-Control (MVC) paradigm. The model in this case is the javax.swing.table.TableModel. The view is presented by the cooperative effort of the HTMLColumnTableModel, the HTMLTableColumn, and HTML components. The HTMLTable is the controller and handles the interaction between the TableModel, the view, and input (such as, settings in an HTML template).

The following is a brief description of the key classes and their roles in generating an HTML table:

**HTMLTable**

This class is the controller of generating the HTML output. It communicates between the TableModel, which has the data, and the HTMLTableColumnModel, the view. The values for attributes for the HTML element, TABLE, are also controlled here.

**TableModel**

Represents the data model for the data to be presented. The columns and the individual cells of data within the columns are controlled here.

**HTMLTableColumnModel**

Represents the view model for the columns that will be presented. The columns and the order of the columns to present from the TableModel are maintained here.

**HTMLTableColumn**

Represents a logical column within the HTMLTable. Sometimes the HTMLTableColumn is generated as more than one HTML column, but the information is presented as a single column with a single header. General attributes of the column can be maintained here. The object to present as the header is controlled here. The ability to associate a custom header component and a custom cell component with the column is also available.

## HTML components

HTML components are used to generate the presentation for the HTMLTable. In particular, a subclass of the HTMLTableHeaderComponent is generally used to present the header, although there are exceptions. Also, a subclass of the HTMLTableCellComponent is generally used to present a cell in the table, although this too has exceptions.

## Initializing the HTMLTable Object

For the HTMLTable to be able to render a table, the following attributes must be set:

- tableModel, which is set using the setTableModel method
- tableColumnModel, which is set using the setTableColumnModel method
- locale, which is set using the setLocale
- outputStream, which is set using the setOutputStream method

After these fields are set, the createDefaultColumnsFromModel method must be called. This method reads the columns from the TableModel, creates an instance of an HTMLTableColumn, and then hands off the new HTMLTableColumn instance to the HTMLTableColumnModel.

Several default constructors perform many of these steps for you. For example, the following constructor:

```
HTMLTable( Vector rowData, Vector columnNames, Locale locale )
```

initializes an instance of the RowDataTableModel, a subclass of TableModel, and an instance of DefaultHTMLTableColumnModel, a subclass of HTMLTableColumnModel. It then calls the createDefaultColumnsFromModel method. The only attribute left to set is the outputStream field.

The code in that constructor follows:

```
RowDataTableModel rowDataTableModel = new RowDataTableModel();
rowDataTableModel.setRowDataObjects(rowData);
rowDataTableModel.setTableColumns(columnNames);
rowDataTableModel.setLocale( locale );
setLocale( locale );
setTableModel( rowDataTableModel );
setTableColumnModel( new DefaultHTMLTableColumnModel() );
createDefaultColumnsFromModel();
```

Rather than pass the HTMLTableColumnModel uninstantiated, as above, the HTMLTableColumnModel class could be passed in already initialized with HTMLTableColumns. In this case, there is no need to call the createDefaultColumnsFromModel method on the HTMLTable instance. More information will be given about this in the section describing the detail of the HTMLTable, later in this chapter. Also, see the Javadoc for the HTMLTable class.

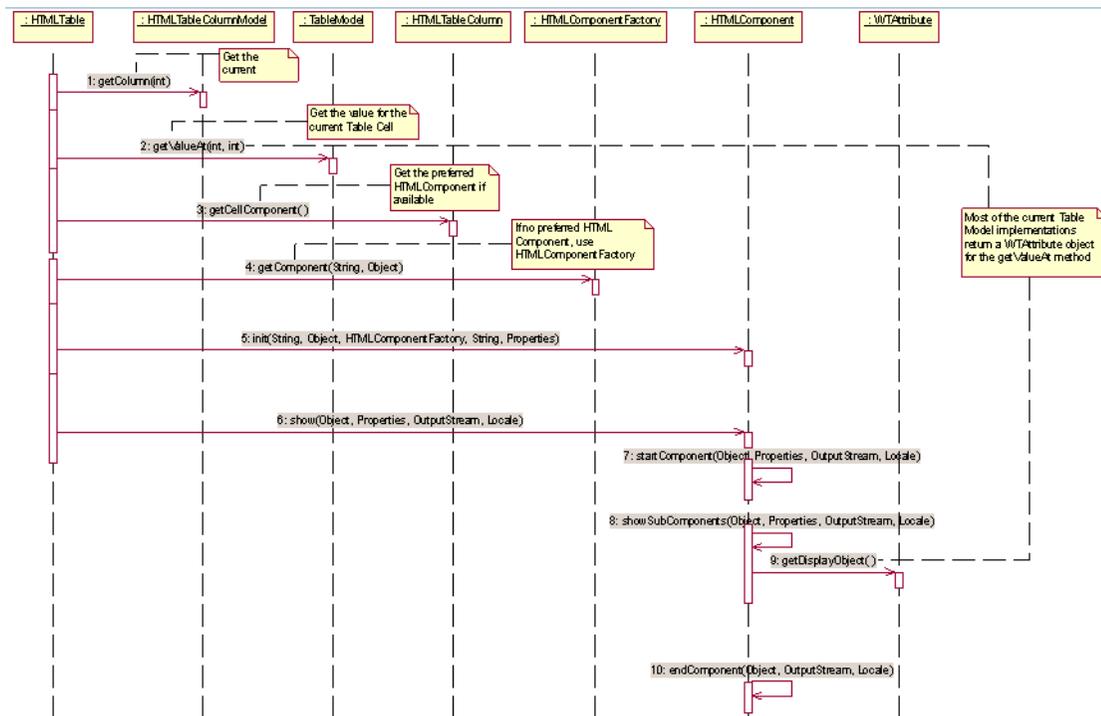
## Rendering the Table

Understanding the process of rendering the table helps understand how to customize the `HTMLTable`'s rendering of the table. Briefly, for each row in the `TableModel` instance, row `i`, all of the columns in the `HTMLTableColumnModel` are looped through one at a time, column `j`. The table cell at position `(i, j)` renders the value returned from the `TableModel` through the call `getValueAt(i, j)` using an `HTML` component.

This process involves selection of the `HTML` component that renders the value at cell `(i, j)`:

1. For each row in the `TableModel` instance, row `i`, all of the columns in the `HTMLTableColumnModel` are looped through one at a time, column `j`.
2. The method `printRow` is called to loop over all of the columns to present in a given row.
3. The `HTMLTableColumn` returned for column `j` references the column `getModelIndex()` in the `TableModel`. The actual call to get the value to display in the table cell is `getValueAt(i, currentColumn.getModelIndex())`.
4. The `HTML` component to render the cell at `(i, j)` is selected through the process.
5. If the current `HTMLTableColumn` returns an `HTML` component from the call `getCellComponent()`, that `HTML` component is used. If a null value is returned, then the `HTMLComponentFactory` is used to select the `HTMLComponent`. The `String` to use as the context for the factory is the field, `cellSelector`. The default value is `"CellComponent"`.

The following is the process of rendering a single cell in the table for position (i, j) within the printRow method:



## HTML Table Service

The HTML table service is a service that is provided to manipulate the HTMLTables through a Windchill script API. This service provides access to the HTMLTable that will render a table through script calls in the HTML template.

### Using the HTML Table Service

To use the HTML table service, you must have an HTMLTable instance and have the HTMLTable instance registered with the HTML Tables service. Once these are done, you can use Windchill script calls to manipulate the HTMLTable.

Depending on the HTMLTable class and the TableModel referenced by the HTMLTable, you have a number of Windchill script calls available.

The following is an example of registering an HTMLTable with the HTML table service from within a TemplateProcessor:

```
ExpandableTable table = new ExpandableTable(resultVector, this,
    locale, os );
table.init(null, null, null, null, null);
table.setOutputStream(os);
table.setLocale(locale);
getHTMLTableService().setHtmlTable(table);
```

The following is the format for all HTML table service calls:

```
<Script language=Windchill>
<!--
tableService action=<Action Name> <Additional Name/Value pairs>
-->
</Script>
```

## Standard Actions

The following are some of the actions that are available through the Windchill scripts:

### **SHOW**

This call invokes the show method on the HTMLTable instance that this service is managing.

### **ADDCOLUMN**

This should be called if your implementation of the Swing TableModel supports adding a new column (the TableModel interface does not define this action).

### **DELETECOLUMN**

This removes the column from the set of columns to display that HTMLTableColumnModel maintains. It does not delete the column from TableModel.

### **MOVECOLUMN**

Exchanges the View position of two columns. The position in the TableModel is not affected, but the position in the HTMLTableColumnModel is. The result is that the positions of the two columns are switched in the visual presentation.

### **SETTABLEATTRIBUTES**

If there are attribute values that you want to pass on to the HTML Table (such as, the TABLE tag attributes), you can do that with this call.

The preceding actions are a subset of those available in the BasicTableService. More specialized services are also available in the AssociationListTableService and the ListContentTableService.

## Using HTML Components

This section gives more detailed instructions on how to use HTML components. For a general overview, see the preceding section on the overview of HTML components, tables, and the table service. For more detailed information on how the functionality for HTML components was actually implemented in Windchill, see the section on HTML Component Implementation, later in this chapter.

Following this section on HTML components and the following sections on HTML tables and the table service is a section describing specifically how to perform a multiselect in the HTML client. This is a common action in HTML clients that you are likely to find very useful.

### The Standard Process

Using an HTML component is based on a two-stage process:

- Calling the init method.

The responsibility of the init method is to set the tag attribute values and to load any subcomponents that should be used to generate output. (For a more detailed description of the parameters and their usage, see the Javadoc.)

- Calling the show method.

This is the method that generates a String of the HTML text to present and returns it. (For a more detailed description of the parameters and their usage, see the Javadoc.)

### The init Method and Cascading

The default sequence of events in the init method is as follows:

1. All of the existing subcomponents are removed.
2. The printTagAttributes field is set by reading from wt.properties. The default value is true. If it is set to false, no defaults are read from wt.properties. The current value of the serviceName attribute is used to locate the entries in wt.properties. If the properties object has a key, "SERVICENAME", then this value is used as the serviceName to get values from wt.properties.
3. The useDefaultsOnly field is set by reading from wt.properties. The default value is false. If it is set to true, then any overriding values passed in the properties object are ignored. If it is set to false, then the properties object is inspected for any overrides of the default values from wt.properties.

An important feature of the init method is the cascading of the serviceName and of default values. The properties object that is passed in can contain the following:

- A new serviceName to be used to retrieve default values from wt.properties
- Settings for the tag attributes values that will override default values, if any

Thus, by passing around the same properties object to a set of HTML components, you can pass a shared serviceName and shared set of tag attribute values. This provides the ability to generate consistent look and feel that can be set dynamically and is able to have the scope of the entire page, a table, or an individual HTML component.

## The show Method

The sequence of events in the show method in the HTMLComponent follows:

1. The startComponent method is called. By default, this call prints out the HTML tag and any attributes that had values defined in the init method.
2. The showSubComponents method is then called. All of the subcomponents that were added to the component will have their show method called. The subcomponents are located in the vector htmlComponents. Using the API call getHtmlComponents(), the vector with any current subcomponents is returned to you.
3. The endComponent method is the last to be called. This method adds the end tag to the return String.

## Using the HTMLComponentFactory

The two aspects to using the HTMLComponentFactory are as follows:

- Property entries create an application context service property mapping a page element or feature to a component class.

For each HTML component that you want to be found, you must place an entry in the htmlcomponent.properties file. The typical entry looks like the following:

```
wt.services/rsc/default/  
  wt.templateutil.components.HTMLComponent/  
<Element>/<Display Class>/0=  
  <Class of HTML Component to be Used>
```

- Use of the HTMLValueObject/WTAttribute.

If the parameter requestorClass in the getComponent call is an instance of HTMLValueObject or WTAttribute, then the getDisplayObject method is called to return the class object that will be used by the factory to locate the desired HTMLComponent.

## Creating a New Base HTML Component

To create a new base component, such as HTMLFontComponent, perform the following steps:

1. Subclass wt.templateutil.components.HTMLComponent.

2. Implement the public `String getTag()` method. This method is used to return the HTML tag that will be used for this HTML element.
3. Add static final Strings that represent the valid attributes for the tag defined in the preceding step.
4. Implement the public void `getTagListArray()` method as follows:

```
public void setTagListArray()
{
    String tagNames[] = {<Array of Static Strings that represent
        tag attributes>};
    setTagList( tagNames );
}

```

5. Implement the `init` method with at least the following:

```
public void init( String selector, Object value,
    HTMLComponentFactory componentFactory, String mode,
    Properties props )
{
    super.init( selector, value, componentFactory, mode, props );
}

```

The `super.init()` call causes the default initialization process in the `HTMLComponent` class to be invoked.

## Creating a Custom HTML Component

A base HTML component prints only the tag, the tag attributes of the HTML component, and any subcomponents that were added to the HTML component. For more specialized behavior, you must override some of the methods of the base HTML component. The following subsections describe some common reasons for doing so and their implementation.

### Adding Subcomponents by Default to Generate Text

You may want to create a component that should have specific subcomponents responsible for generating the text between the HTML element tags. An example of this is presenting text with a standard icon in the cell of an HTML table. You could have a special HTML component that generates the link to the icon. This would encapsulate the knowledge of generating this link. Then, you could have the HTML component that generates the cell in the table use this component.

In this case, you would want to choose which base HTML component to subclass and then override the `init` method. Within the `init` method you call `super.init(...)` as usual, but you must also instantiate and initialize the HTML subcomponents.

For the example just discussed, the `IconFirstCellComponent` is the subcomponent that would be used. This HTML component always has two subcomponents. The first subcomponent is responsible for generating the icon for a business object. The second subcomponent is responsible for generating the Number of the business object and a link to the business object's `ObjectProperties` page. The code follows:

```
public void init( String selector, Object value,
    HTMLComponentFactory componentFactory, String mode,
    Properties props ) {

    super.init( selector, value, componentFactory, mode, props );

    if ( props == null )
    {
        props = new Properties();
    }

    DefaultHTMLCellComponent cellComponent1 =
        new DefaultHTMLCellComponent();
    cellComponent1.init( selector, null, componentFactory, mode,props );
    DefaultHTMLCellComponent cellComponent2 = new
        DefaultHTMLCellComponent();
    cellComponent2.init( selector, null, componentFactory, mode,props );

    DefaultIconComponent iconComponent = new DefaultIconComponent();
    iconComponent.init(selector, null, componentFactory, mode,props);
    cellComponent1.getHtmlComponents().addElement( iconComponent );

    DefaultLinkComponent linkComponent = new DefaultLinkComponent();
    DefaultStringComponent stringComponent = new DefaultStringComponent();
    linkComponent.getHtmlComponents().addElement( stringComponent );
    cellComponent2.getHtmlComponents().addElement( linkComponent );
    getHtmlComponents().addElement( cellComponent1 );
    getHtmlComponents().addElement( cellComponent2 );
}
```

In the preceding code, the same properties object is passed to the subcomponents as is used by the parent component. This allows the cascading of the `serviceName` and any default values for the tag attributes.

## Overriding the showSubComponent Method to Display Text Directly

At some point, you will want an HTML component that displays text. The correct way to implement this is to override the `showSubComponent` method. The `showSubComponent` method is responsible for generating the text that appears between the starting HTML tag and the ending HTML tag. A sample override of the `showSubComponent` method follows. One technique displayed is how to deal with a `WTAttribute` instance that is passed to the HTML component. The `objectPropertyValue` and the `objectPropertyName` Windchill script calls pass in a `WTAttribute` (for further information, see the section on changing the presentation of attributes in the HTML client). The `HTMLTables` also pass in a `WTAttribute` or `HTMLValueObject` most of the time.

```
public String startComponent( Object value, Properties formData,
    OutputStream os, Locale locale )
{
    Object imageSource = value;
    if ( value instanceof WTAttribute )
    {
        imageSource = ((WTAttribute)value).getSource();
    }
    try
    {
        setTagValue("SRC", getObjectIconImgTag(
            (wt.fc.WTObject)imageSource ) );
    }
    catch (wt.util.WTException wte)
    {
        setTagValue("SRC", "wt.util.WTException while generating
            icon" );
    }
    String resultStr = super.startComponent( value, formData, os,
        locale );
    return resultStr;
}
```

In this example, the `getObjectIconImgTag` method is also implemented in this class and knows how to retrieve the correct icon based on the object and its current state, that is, whether it is checked out or not.

## Adding Logic to the Generation of the Tag Attributes

You may want to add some logic, probably based on a business object, to the setting of a tag attributes value. An example of this is the `DefaultIconComponent`. In this component, the value of the SRC attribute is set based on the class and state of a business object. It is not possible to read this value from `wt.properties`. To implement this type of logic, you must override the `startComponent(...)` method in the HTML component. There you would override the default settings. The following is an example:

```
public String startComponent( Object value, Properties formData,
    PrintWriter out, Locale locale )
{
    try
```

```

    {
    setTagValue("SRC", getObjectIconImgTag( value ) );
    }
    catch (wt.util.WTException wte)
    {
    setTagValue("SRC", "wt.util.WTException while generating icon" );
    }
    String resultStr = super.startComponent( value, formData,
    out, locale );
    return resultStr;
    }
}

```

In this example, the `getObjectIconImgTag` call returns a `String` with the URL of the desired icon. Note the call to the `super.startComponent(...)`. After resetting the value of the tag attribute, the call to the `super` results in using the `HTMLComponent`'s implementation of `startComponent` that will enumerate through the tag attributes that are set and present them.

## Setting Tag Attributes Directly

Sometimes in your custom HTML component, you want to set the value of a tag attribute directly and override the default values. There are two ways to do this:

- Using an API call to set an individual tag attribute value.

To set the value on an individual tag attribute value, you can call the `setTagValue(...)` method either from within the `init(...)` method or on the HTML component directly. If you call the `setTagValue` from within the `init` method, you must call it after calling `super.init(...)`. Otherwise, the default values may overwrite your custom setting. The following is sample code:

```

public void init( String selector, Object value,
    HTMLComponentFactory componentFactory, String mode,
    Properties props )
    {
    super.init( selector, value, componentFactory, mode,
    props);
    setTagValue( HTMLFontComponent.COLOR,
    BasicTemplateProcessor.getWCColor("f-label"));
    }
}

```

You can also call the `setTagValue` directly on the HTML component. You should do this only after the `init` method is called. The following is some sample code:

```

HTMLFontComponent newFont = new HTMLFont();
newFont.init(selector, value, componentFactory, mode, props );
newFont.setTagValue( HTMLFontComponent.COLOR,
    BasicTemplateProcessor.getWCColor("f-label"));
}

```

- Passing in values on the properties object passed into the `init` method (overrides default values)

This technique can be used to set the value on an individual HTML component or to have the settings cascade through the subcomponents (for further information, see the preceding section on the init method and cascading). It also allows you use the same properties object across several HTML components.

The following is sample code:

```
if ( props == null )
{
  props = new Properties();
}
// This is not the ideal use of Strings. This is done for
// demonstrative purposes
props.put( "TD" + "." + HTMLTableCellComponent.BGCOLOR ,
  BasicTemplateProcessor.getWCColor(t1-bg-evenrow" ) );

DefaultHTMLCellComponent cellComponent1 = new
  DefaultHTMLCellComponent();
cellComponent1.init( selector, null, componentFactory, mode,
  props );
DefaultHTMLCellComponent cellComponent2 = new
  DefaultHTMLCellComponent();
cellComponent2.init( selector, null, componentFactory, mode,
  props );
```

The syntax used for specifying tag attribute values passed on the properties object differs from that used for the direct method. When passing in values in the properties object through the init method, use the following syntax:

**props.put(<Element tag>.<Attribute Name>,<Value>, );**

For example, the following is a valid entry:

```
props.put( "TD.BGCOLOR" ,
  BasicTemplateProcessor.getWCColor( "t1-bg-evenrow" ) );
```

Many of the base HTML components have convenience static Strings to deal with this syntax. Following is an example:

```
props.put( HTMLTableCellComponent.TD_ALIGN , "RIGHT" );
```

Note, however, that there is not full support of these convenience Strings.

## Selecting a New Context(#Service Name) for Initializing the Default Tag Attributes

The default tag attribute values in wt.properties look like the following:

```
wt.templateutil.component.<Tag>.<Attribute Name>=<Value>
```

An example of this syntax follows:

```
wt.templateutil.component.TD.ALIGN=Left
```

These are the entries with which all HTML components are initialized.

To switch to a different set of default values for your HTML component, you must set the value of the serviceName field. There are two ways to change the serviceName so that a custom set of initial values are set: directly setting the new service name and cascading the new service name.

### Directly Setting the New Service Name

You can call the method setServiceName directly. You must do this before calling the init method if you want to use the custom initial values. The following code is an example:

```
HTMLTableCellComponent myCellComponent =  
    new HTMLTableCellComponent();  
myCellComponent.setServiceName( "CustomService" );  
myCellComponent.init( ... );
```

The HTML component myCellComponent looks in wt.properties for entries of the following form:

```
CustomService.<Tag>.<Attribute Name>=<Value>
```

### Cascading the New Service Name

Using the cascading feature of HTML components, you can pass in the new service name in the properties object passed into the HTML component in the init method. The following is example code:

```
HTMLTableCellComponent myCellComponent =  
    new HTMLTableCellComponent();  
Properties customProperties = new Properties();  
customProperties.put( HTMLComponent.SERVICENAME ,  
    "CustomService" );  
myCellComponent.init(selector, value, componentFactory, mode,  
    customProperties );
```

The HTML component myCellComponent will look in wt.properties for entries of the following form:

```
CustomService.<Tag>.<Attribute Name>=<Value>
```

## Retrieving the Default Service Name

If the `serviceName` is not set on the HTML component or passed in the `properties` parameter of the `init` call, the default value of `wt.templateutil.component` is used. If you want your HTML component to ensure it is always using the default value, the default value can always be retrieved as the following:

```
String defaultContext = wt.templateutil.components.HTMLComponent.  
    DEFAULT_SERVICE_NAME;
```

## Printing Tags Only

You may want to print the tag for the HTML elements only and not the tag attributes. For the given service name of your component, you add the following line in `wt.properties` to affect the printing of the tag attributes:

```
<Service Name>.PRINT_TAG_ATTR=true
```

Note that the default setting for all components is `true` unless overridden.

## Printing the Default Tag Values Only

You may not want the default values of the tag attributes overridden. In this case, place the following entry in `wt.properties`, again using the service name of your component:

```
<Service Name>.DEFAULTS_ONLY=true
```

Note that the default setting for all components is `false` unless overridden.

## Using HTML Tables and Custom Tables

This section gives more detailed instructions on how to use HTML tables and custom tables. For a general overview, see the section earlier in this chapter on the overview of HTML components, tables, and the table service. For more detailed information on how the functionality for HTML tables was actually implemented in Windchill, see the section on HTML Table Generation and Table Service Implementation, later in this chapter.

Following this section on HTML tables and the following sections on the HTML table service is a section describing specifically how to change the presentation of attributes in the HTML client. This is a common action in HTML clients that you are likely to find very useful.

The `HTMLTable` is the base class that generates tables from implementations of `TableModels`. It works with the `HTMLTableColumns` and an implementation of an `HTMLTableColumnModel` to control the generation of the HTML table.

There are several ways to customize and/or configure the `HTMLTable` to control the presentation of the table that is generated by the `HTMLTable`. The following three ways are currently available:

- Configuring the `HTMLTable` through fields on the `HTMLTable` as follows:
  - Setting the `cellSelector` to look for custom HTML components to render the cell.
  - Setting the `headerSelector` to look for custom HTML components to render the header.
  - Setting the `displayHeader` field to show or hide the table headers.
- Overriding behavior by subclassing the following:
  - `createDefaultColumnsFromModel`
  - `printHeaders`
  - `printRow`
- Using the HTML table service

### Printing a Standalone HTML Table

#### The Basic Process

The process of printing a stand-alone HTML table (versus a table within another HTML component, as shown later in this section) is as follows:

1. Provide an implementation of the `javax.swing.table.TableModel` class.
2. Provide an implementation of the `wt.templateutil.table.HTMLTableColumnModel`.

3. Provide an implementation of the HTMLTable and pass in the TableModel and the HTMLTableColumnModel:
  - a. Set the Locale.
  - b. Set the OutputStream.
  - c. Call the createDefaultColumnsFromModel (only if the constructor you call does not do it).
  - d. Call the init method with all null values, that is:
 

```
init( null, null, null, null, null );
```
4. Call the show( Properties formData ) method.

This process can be done by performing individual method calls or using a constructor that does several of the preceding steps for you.

## Examples

The following is an example of initializing an instance of wt.enterprise.table.WThtmlTable and printing out the table in a step by step manner:

```
HTMLTable table = new HTMLTable();
RowDataTableModel tableModel = new RowDataTableModel();
tableModel.setRowDataObjects( resultVector ); //List of Windchill
// Business objects that represent the rows
tableModel.setTableColumns( namesVector ); // List of
// names(attributes) that represent the columns
table.setTableColumnModel(new DefaultHTMLTableColumnModel() );
table.setTableModel(tableModel);
table.createDefaultColumnsFromModel();
table.setOutputStream( os );
table.setLocale( locale );
table.init( null, null, null, null, null ); // For the presentation
// of a stand alone table, the init does not do anything
table.show();
```

The following is an example of initializing an instance of wt.enterprise.table.WThtmlTable and printing the table using one of the HTMLTable constructors to handle most of the initialization:

```
HTMLTable table = new HTMLTable(resultVector, namesVector, locale);
table.setOutputStream( os );
table.init( null, null, null, null, null ); // For the presentation
// of a stand alone table, the init does not do anything
table.show();
```

In this case, the constructor takes care of initializing a RowDataTableModel (with the resultVector and the namesVector), setting the locale, and calling createDefaultColumnsFromModel to initialize the DefaultHTMLTableColumnModel.

## Setting cellSelector to Select Custom HTML Components for the Cells

In the process of printing the table, each cell in the body of the table is rendered by an HTML component. The HTML component comes from either the HTMLTableColumn for that column in the table or from the HTMLComponentFactory (if the HTMLTableColumn does not specify an HTMLComponent). By default, the HTMLComponentFactory looks for properties of the following form:

```
wt.services/rsc/default/wt.templateutil.components.HTMLComponent/  
CellComponent/<Class of Display Object>/0=<HTML component>
```

If you want to specify a particular set of HTML components to be used for a specific HTMLTable to generate the cells, you can set the cellSelector field and the new value will be used by the HTMLComponentFactory to select the HTML components. For example, assume you make the following call:

```
table.setCellSelector("ChangeComponents");
```

Then the HTMLComponentFactory looks for properties with the following syntax:

```
wt.services/rsc/default/wt.templateutil.components.HTMLComponent/  
ChangeComponents/<Class of Display Object>/0=<HTML component>
```

## Enabling or Hiding Table Headers

You may not want headers to be presented on a particular table. To hide/disable the headers, you simply set the value on the displayHeader field. This is a boolean attribute and it can be set with the following call:

```
table.setDisplayHeader( true/false );
```

## Using Custom Columns by Overriding createDefaultColumnsFromModel

The main reason to override the createDefaultColumnsFromModel method is to control which subclass of the HTMLTableColumn is used in the DefaultHTMLTableColumnModel and how the columns are initialized. An example of customized behavior is WTHtmlTable. The ability to add a column of checkboxes to the table's presentation is enabled in createDefaultColumnsFromModel.

## Customizing the Table Body Generation by Overriding printRow

The simplified, default process of printing the body of the table is as follows:

1. For each row in the TableModel, row *i*, call the printRow method in the HTMLTable.
2. In the printRow method, loop over the HTMLTableColumns in the HTMLTableColumnModel, column *j*:
  - a. Call the getValueAt(*i*, *j*) to get the object to display at position (*i*,*j*).
  - b. See if the HTMLTableColumn returns an HTML component from the call, getCellComponent(). If so, use this component to render the cell. Otherwise, call the HTMLComponentFactory with the current value of cellSelector and the object from the preceding step.
  - c. Call the init and the show on the HTML component

You can customize the generation of a row in the table in several ways. The most common is to have a more complex row generated. In most of the tables presented in Windchill, each row in the table presents a single business object. The default presentation is a single row of cells where each cell presents an attribute. The following figure shows an example:

Number	Name	Quantity	Unit	Described By	Latest Version
 <a href="#">9123630724</a>	TIMING_COVER_LWR	1.0	ea		<a href="#">A</a> <a href="#">All Versions</a>
 <a href="#">7030569361</a>	TIMING_PLATE	1.0	ea		<a href="#">A</a> <a href="#">All Versions</a>
 <a href="#">1094809962</a>	TIMING_COVER_UP	1.0	ea		<a href="#">A</a> <a href="#">All Versions</a>
 <a href="#">7135138263</a>	TIMING_COVER_BUSHING	4.0	ea		<a href="#">A</a> <a href="#">All Versions</a>

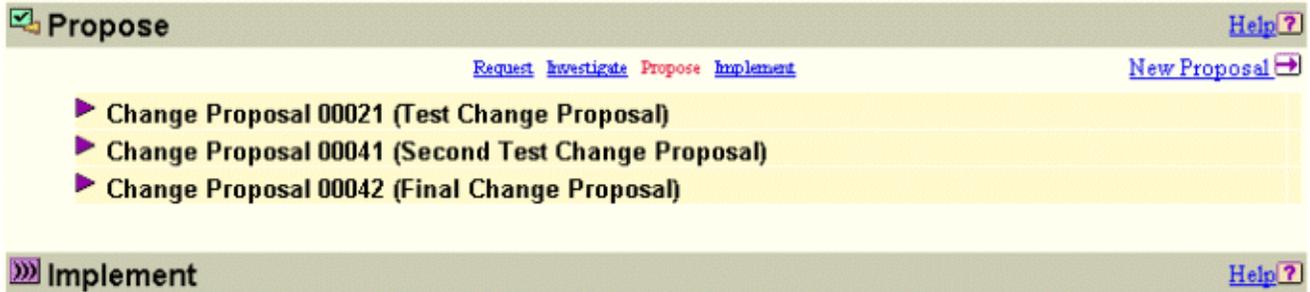
You may want a richer presentation of the business object. For example, you might want to include a description of the object. This would be best presented on its own row. The following figure is an example of such a table:

Relevant Parts and Documents					<a href="#">Add Change</a>
Number	Name	Version	Type	Source	
 <a href="#">1</a>	EGadWork	A	Document		
<b>Comments:</b> This is some superfluous text					
 <a href="#">10</a>	SWidWork	A	Document		
<b>Comments:</b>					
 <a href="#">9</a>	EWidWork	A	Document		
<b>Comments:</b>					
 <a href="#">1255875099</a>	L25_WATER_PUMP	A	Part	Make	
<b>Comments:</b>					

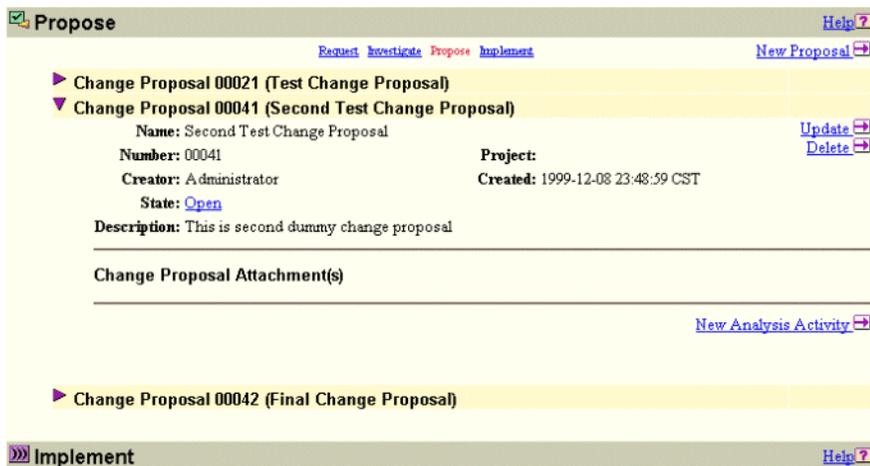
To create this second table, the printRow method was overridden so that each logical row in the TableModel resulted in two physical rows in the generated table. The essential part of the code is as follows:

```
public String printRow( int rowNumber, Object value,
    Properties formData )
{
    super.printRow(rowNumber, value, formData);
    PrintWriter print_writer = getPrintWriter(getOutputStream(),
        getLocale() );
    print_writer.print( super.printRow (rowNumber, null, formData) );
    print_writer.println("<TR>");
    // Customized second row of the logical row
    ...
    // Now close the row
    print_writer.println("</TR>");
    print_writer.flush();
}
```

An even more sophisticated use of overriding the printRow method is to allow expandable rows in the HTML table. An example of this is the Change Management Change Request page. This page presents a table of Change Proposals initially in the standard form, as shown in the following figure:



If you would like to see more information of the 00041 change proposal, you might click on the purple arrow to the right of the Change Proposal to expand the change proposal. The resulting page looks like the following figure:



The code to provide this expand/contract ability has the following basic form:

```
public String printRow( int rowNum, Object value,
    Properties formData )
{
    super.printRow(rowNumber, value, formData);
    PrintWriter print_writer = getPrintWriter(getOutputStream(),
        getLocale() );
    print_writer.print( super.printRow (rowNumber, null, formData) );
    Boolean expand = new Boolean( getFormData().getProperty( EXPAND ) );
    if (expand.booleanValue() )
    {
        print_writer.println("<TR>");
        // Customized second row of the logical row
        // In this case, processing a subtemplate is done to generate the
        // values.
    }
}
```

```

...
// Now close the row
print_writer.println("</TR>");
}
        print_writer.flush();
}

```

## Customizing Header Generation by Overriding printHeaders

The simplified, default process of generating the headers for a table is as follows:

1. Loop over the HTMLTableColumns in the HTMLTableColumnModel:
  - a. See if the HTMLTableColumn has a value for the headerObject. If not, call the TableModel's getColumnName method for the object to present in the header.
  - b. See if the HTMLTableColumn returns an HTML component from the call getHeaderComponent(). If so, use this component to render the header. Otherwise, call the HTMLComponentFactory with the current value of headerSelector and the object from the preceding step.
  - c. Call the init and the show on the HTML component.

To override this process, subclass the HTMLTable and override this method. Then you can select which part of the process to override. It might be that you have a completely different scheme to generate the headers. You might want to do something as simple as have the number of the column as the header. This could be done easily by overriding the printHeaders method.

## Access the contextObj and TemplateState in an HTML Component in the Table

At times, to present the information required in the cell in the table, you need not only the object represented in the current row, but also the contextObj of the current HTML page in which the table is be presented. In this case, you must have access to the current contextObj and/or the HTTPState object of the current TemplateProcessor.

This information can be accessed in the HTML component that is used by the HTMLTable object. To access this information, refer to the following sample code:

```

public String startComponent( Object obj, Properties formData,
    OutputStream os, Locale locale )
{
    StringBuffer startStr = new StringBuffer(50);

    startStr.append( "<" );
    startStr.append( getTag() );
    startStr.append( " " );

    Object contextObj = null;

    String contextID = formData.getProperty( PageContext.KEY );

```

```

        if ( contextID != null && PageContext.getContext( contextID )
            != null )
        {
            PageContext context = PageContext.getContext( contextID );
            HTTPState state = (HTTPState)context.get( HTTPSTATE );
            contextObj = state.getContextObj();
        }
        else
        {
            if (VERBOSE)
            {
                System.err.println("DefaultLinkComponent -
                    startComponent :
                    No page context ");
            }
        }
    }
    ...

```

## Selecting a New Service Name for Custom Default Tag Attributes

This example is performed just like the HTML components (mainly because the HTMLTable is an HTML component). The serviceName field of the HTMLTable is used to specify which block of entries in the wt.properties file to read to initialize the tag attribute values. By default, the HTMLTable passes in its serviceName value to all of the HTML components used to render the headers and the cells through the Properties object passed in the init method of the HTML component. Consequently, all of the HTML components have access to the preferred set of default values to be used in the table. By setting the serviceName of the HTMLTable, you have the headers and cells use the same serviceName. Therefore, they all go to the same set of default values and give custom look to the table.

## Registering for the HTML Table Service

The wt.templateutil.table.TemplateProcessorTableService can be used for manipulating the HTMLTable through Windchill script calls. By registering your HTMLTable with this service, you can make Windchill script calls to affect this table's presentation. The class wt.enterprise.BasicTemplateProcessor provides a method for getting and setting the aggregated instance. The following is sample code. This code assumes that you are working in a subclass of BasicTemplate.

```
getHTMLTableService( ).setHTMLTable(table );
```

After this, your HTMLTable can be manipulated by Windchill script calls using the API mentioned above. To show your HTMLTable, use the Windchill script call with action=show.

Once you have registered for this service, you have a number of actions you can take on the table to modify its presentation. These will be discussed in the following section on the HTML table service.

# Using the HTML Table Service

## Overview

This section gives more detailed instructions on how to use the HTML table service. For a general overview, see the section earlier in this chapter on the overview of HTML components, tables, and the table service. For more detailed information on how the functionality for the HTML table service was actually implemented in Windchill, see the section on HTML Table Generation and Table Service Implementation, later in this chapter.

The HTML table service is essentially a Windchill script API to affect the presentation of HTML tables. The Windchill script call always has the following format:

```
tableService action=<Name of Action> <Additional Name/Value Pairs>
```

The service is an event-based service that has its listeners registered through `wt.properties`. More listeners (that is, services) can be created and registered.

Several services are available in Windchill currently. The following three are discussed in this section:

- `BasicTableService`
- `AssociationListTableService`
- `ListContentTableService`

## BasicTableService

This service represents the basic services that you would want for most HTMLTables. There might be some additional interfaces that you need to implement in either your `HTMLTable` subclass or your `TableModel` implementation for some of the actions. Default implementations of these interfaces are available currently.

All of the services available in the `BasicTableService` and the interfaces required for some of the services are not discussed in this manual. For a more detailed description of the Windchill script calls and the interfaces implementations required to use them, see the Javadoc for these classes. This section simply lists the actions that are available:

### SHOW

This call invokes the `show` method on the `HTMLTable` instance that this service is managing.

### ADDCOLUMN

You can call this if your implementation of the `Swing TableModel` supports adding a new column (the `TableModel` interface does not define this action).

**DELETECOLUMN**

This removes the column from the set of columns to display that HTMLTableColumnModel maintains. It does not delete the column from TableModel.

**MOVECOLUMN**

Switches the position of a column within the HTMLTableColumnModel's presentation order. It does not affect the TableModel.

**ADDCOLUMNS**

This is similar to ADDCOLUMN except that multiple columns are added at once.

**DELETECOLUMNS**

This is similar to DELETECOLUMN except that multiple columns are deleted at once from the HTMLTableColumnModel's presentation list.

**SETCOLUMNATTRIBUTES**

If there are attribute values you want to pass on to all of the HTML Table cells (that is, the TD tag attributes) within a given column, use this call.

**SETHEADERATTRIBUTES**

If there are attribute values you want to pass on to all of the headers of a column (that is, the TH tag attributes) within a given column, use this call.

**SETTABLEATTRIBUTES**

If there are attribute values you want to pass on to the HTML Table (that is, the TABLE tag attributes), use this call.

**SETSERVICENAME**

Sets the service name that is used by the HTMLTable, the HTML components used to render the table header, and the HTML components used to render the table cells to initialize the values from wt.properties.

**SETCELLCOMPONENT**

Sets the HTML component to be used to generate the HTML for each cell in the column.

**SETHEADERCOMPONENT**

Sets the HTML component to be used to generate the HTML for the header of the column.

**HIDETABLEHEADER**

Sets the displayHeader field on the HTMLTable. It can be set to either true or false.

**SETHEADERFROMRESOURCE**

Sets a resource bundle and a key in that resource bundle that define the localized text to display in a column.

**SETHEADERFROMTARGETCLASS**

Introspection is used sometimes to generate the header. In this case, you might want to force the class that the introspection is performed against.

## AssociationListTableService

This service is an implementation of an HTML table service that provides special support for working with an AssociationTableModel and the WTHtmlTable.

The primary function of this service is to allow the specification of an association, perform the navigation, and present the results of that navigation in a table.

The type of navigation to be done can be specified in the Windchill script calls. Columns to present for both BinaryLinks and the OtherSide objects can be specified in the Windchill script. The navigation can be performed in one of two ways: a direct call to a navigation can be done or a subclass of the BaseQueryService can be used to perform the navigation. The direct call to a navigation is the default unless a reference to a subclass of the BaseQueryService is passed in the initAssociationNavigation call.

The following is an example of using the AssociationListTableService that does a navigation directly and presents the results:

```
<SCRIPT LANGUAGE= Windchill>
<!--
tableService action=initAssociationNavigation ROLE=uses
  LINKCLASSNAME=wt.part.WTPartUsageLink
  OTHERSIDECLASS=wt.part.WTPartMaster
tableService action=initAssociationTable othersideattributes=name,
  number,type linkattributes=modifyTimestamp,quantity
tableService action=setHeaderAttributes COLUMNNUMBER=ALL th.ALIGN=left
  recurse.font.COLOR="wt.enterprise.BasicTemplateProcessor
  getWCColor styleClass="t1-f-col-head" quotes=none font.size=1
tableService action=setColumnAttributes name=number,quantity
  td.ALIGN=left nowrap font.size=5
tableService action=setTableAttributes table.ALIGN=center
  table.WIDTH=90% table.cellspacing=2 table.cellpadding=2 table.border=4
tableService action=show
-->
</SCRIPT>
```

Note that each line of the Windchill script should begin with "tableService". If you cut and paste the preceding code without removing the additional line wraps, you will get exceptions. In particular, the lines that begin with "OTHERSIDECLASS" and "linkattributes" are wrapped lines.

The following is an example of using the AssociationListTableService that uses a subclass of BaseQueryService -- RelevantChangeables specifically -- to perform the navigation. A UML diagram of the RelevantChangeables is presented below the Script calls.

```
<SCRIPT LANGUAGE=Windchill>
<!--
tableService action=initAssociationNavigation BASEQUERYSERVICE=
  RelevantChangeables ROLE=theChangeable2
  LINKCLASSNAME=wt.change2.RelevantAnalysisData
  OTHERSIDECLASS=wt.change2.
tableService action=setAssociationTable ASSOCIATIONTABLE=
  wt.change2.htmlclient.
```

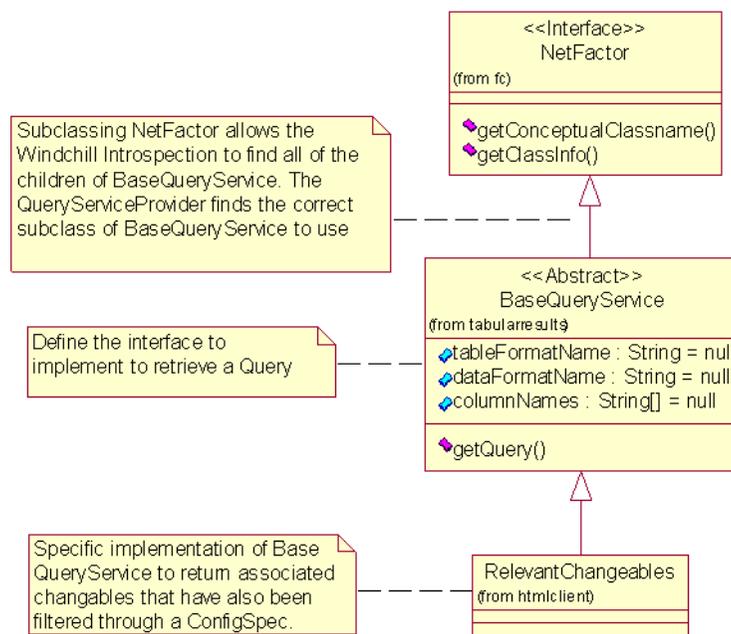
```

tableService action=initAssociationTable OtherSideAttributes=number,
  name,versionInfo,type,
tableService action=setHeaderAttributes COLUMNNUMBER=ALL th.ALIGN=left
  recurse.font.COLOR="wt.enterprise.BasicTemplateProcessor
  getWCColor styleClass="t1-f-col-head" quotes=none font.size=2
tableService action=setColumnAttributes COLUMNNUMBER=ALL td.ALIGN=left
  nowrap font.size=2
tableService action=setTableAttributes table.ALIGN=left table.WIDTH=90%
  table.cellspacing=2 table.cellpadding=2
tableService action=show
-->
</SCRIPT>

```

As before, note that each line of the Windchill script should begin with "tableService". If you cut and paste the preceding code without removing the additional line wraps, you will get exceptions. In particular, the lines that begin with "ROLE" and "table." are wrapped lines.

The following figure is a UML diagram showing the structure of the BaseQueryService. The QueryServiceProvider can locate a modeled subclass of BaseQueryService using the name of the class only. If you have not modeled the class, the QueryServiceProvider requires the fully qualified class path for the class.



The actions supported in the AssociationListTableService are as follows:

#### INITASSOCIATIONNAVIGATION

Takes in the parameters that define an Association navigation and performs the navigation.

### **INITASSOCIATIONTABLE**

Initializes an instance of WTHtmlTable with an instance of an AssociationTableModel. The AssociationTableModel is initialized with the BinaryLink objects that were returned from the association navigation.

### **SETASSOCIATIONTABLEMODEL**

Sets the TableModel that will be used to initialize the subclass of WTHtmlTable used to generate the table. The TableModel has to be a subclass of the AssociationTableModel.

### **SETASSOCIATIONTABLE**

Sets the HTMLTable subclass to be used to generate the table. Currently, a subclass of WTHtmlTable is required.

## **ListContentTableService**

This service is an implementation of an HTML table service that provides special support for presenting the contents of a ContentHolder. The content is retrieved from the current context object of the TemplateProcessor instance making the Windchill script calls. For example, if you are looking at the properties page for a WTPart, the context object is the WTPart whose properties are being displayed. This is also the object from which the list of content is retrieved.

The primary function of this service is to allow the retrieval of the contents -- both UrlData and ApplicationData -- and present those two sets of contents in two tables. One table is for the UrlData and one table is for the ApplicationData. The presentation and display characteristics of the tables are controlled separately. The tables can look the same or they can look different. The tables can be presented one after the other or in different parts of the page.

**Note:** Currently, AggregateData is not supported.

To use ListContentTableService, make the following Windchill script call:

```
tableService action=initializeContents
```

After this call, the URLData contents and the ApplicationData contents are available to be displayed. There is a call to initialize the HTMLTable object for each type of content. This allows the two tables to be controlled, both in style and position, separately.

For URLData, the following call initializes a WTHtmlTable to present the URLData:

```
tableService action=initURLDataTable URLDATAATTRIBUTES=  
urlLocation, description
```

For ApplicationData, the following call initializes a WTHtmlTable to present the ApplicationData:

```
tableService action=INITAPPLICATIONDATATABLE  
APPLICATIONDATAATTRIBUTES=fileName,format,fileSize,  
modifyTimestamp,createdBy
```

Once these WTHtmlTables are created, the rest of the HTML table services are available to customize them, as shown below.

The following is an example of using the ListContentTableService:

```
<SCRIPT LANGUAGE=Windchill>
<--
tableService action=initializeContents // Required call to get Contents
tableService action=initURLDataTable URLDATAATTRIBUTES=urlLocation,
    description
tableService action=setHeaderAttributes name=ALL font.size=2 th.align=left
tableService action=setColumnAttributes name=ALL font.size=2
    recurse.td.bgcolor="wt.enterprise.BasicTemplateProcessor
    getWCColor styleClass=bg-hilite1" quotes=none
tableService action=setTableAttributes table.width=100%
tableService action=show
-->
</SCRIPT>
<SCRIPT LANGUAGE=Windchill>
<!--
tableService action=INITAPPLICATIONDATATABLE APPLICATIONDATAATTRIBUTES=
    fileName,format,fileSize,modifyTimestamp,createdBy
tableService action=setHeaderFromResource POSITION=4 RESOURCEBUNDLE=
    wt.enterprise.enterpriseResource RESOURCEKEY=UPDATED_BY
tableService action=setHeaderAttributes name=ALL font.size=2 th.align=left
tableService action=setColumnAttributes name=ALL font.size=2
    recurse.td.bgcolor="wt.enterprise.BasicTemplateProcessor
    getWCColor styleClass=bg-hilite1" quotes=none
tableService action=setTableAttributes table.width=100%
tableService action=show
-->
</SCRIPT>
```

Note that each line of the Windchill script should begin with "tableService". If you cut and paste the preceding code without removing the additional line wraps, you will get exceptions. In particular, the lines that begin with "APPLICATIONDATAATTRIBUTES" and "RESOURCEBUNDLE" are wrapped lines.

The following actions are supported in the AssociationListTableService:

#### **INITIALIZECONTENTS**

This call takes the current context object of the template processor and, if the context object is a context holder, the ApplicationData and the URLData are retrieved.

#### **INITAPPLICATIONDATATABLE**

This call initializes an instance of a WTHtmlTable with the RowDataTableModel and the ApplicationData. Windchill script calls can now be made to affect the presentation of the table. This call should be made only after the INITIALIZECONTENTS action is called.

**INITURLDATATABLE**

This call initializes an instance of a WTHtmlTable with the RowDataTableModel and the URLData. Windchill script calls can now be made to affect the presentation of the table. This call should be made only after the INITIALIZECONTENTS action is called.

## Performing a Multiselect in the HTML Client

A multiselect refers to the following sequence of actions:

1. Perform a search for candidates (for example, through a local search or navigation).
2. Select any number of the resulting objects. In the HTML client, this is done using checkboxes.
3. Perform some action on all of the selected objects.

This section describes how to present the results of the search or navigation in an HTML table with checkboxes and how to process the HTTP Post that is returned from the HTML form generated from the search or navigate.

### Generating a Column of Checkboxes

The `CheckBoxTableColumn` and `CheckBoxCellComponent` classes can be used to generate the column in a table that presents checkboxes. The `CheckBoxTableColumn` has an instance of `CheckBoxCellComponent` returned from the `getCellComponent`. This results in the `CheckBoxCellComponent` being used to render the cells in the column represented by the `CheckBoxTableColumn`. The `CheckBoxTableColumn` does not have any `headerComponent` set or any default value for the header object.

No header is defined by the `CheckBoxTableColumn`. If any value for the header is to be displayed, it must be done by a Windchill script command or by the HTML component selected by the `HTMLTable` to generate the header.

The cells in the column are generated by the `CheckBoxCellComponent`. The `CheckBoxCellComponent` generates HTML Form input fields with a specific syntax for the name of the input field and the value of the input field. The HTML generated for each cell in the column of checkboxes has the following format:

```
<INPUT VALUE="value " TYPE="CHECKBOX" NAME="name ">
```

Within this format, "*value* " is formatted as follows:

```
<HTML Table Context><CheckBoxCellComponent.CONTEXT_SEPARATOR>  
<OID of object in Row
```

An example of the default form of the value attribute is as follows:

```
HtmlTable_VR:wt.doc.WTDocument:212401
```

In this example, `<HTML Table Context>` is `HtmlTable`; `<CheckBoxCellComponent.CONTEXT_SEPARATOR>` is the underscore character; and `<OID of object in Row>` is `VR:wt.doc.WTDocument:212401`.

Within this format, "*name* " is formatted as follows:

```
checkbox_<CheckBoxCellComponent.CONTEXT_SEPARATOR><OID of object in  
Row>
```

An example of the default form of the name attribute is as follows:

```
checkbox__VR:wt.doc.WTDocument:130918
```

In this example, `<CheckBoxCellComponent.CONTEXT_SEPARATOR>` is the underscore character and `<OID of object in Row>` is `VR:wt.doc.WTDocument:130918`.

The `FormTaskDelegate` that is part of this example shows how to handle the return data so that the relevant information is retrieved. Also note that the object passed in (or the source of a `WTAttribute`) to the `CheckBoxCellComponent` must be persistable if an `OID` is to be generated.

## WTHtmlTable and Enabling the Checkbox

The `WTHtmlTable` is a subclass of `HTMLTable` that can add a column of checkboxes to the presentation of a table by setting the value of the field `presentCheckBox`. The boolean attribute, `presentCheckBox`, indicates if the `WTHtmlTable` should add a column of checkboxes to the presentation of the table. The value of the `presentCheckBox` field when the `createDefaultColumnsFromModel` method is called determines whether a column of checkboxes is added or not. Changing the value of `presentCheckBox` after the `createDefaultColumnsFromModel` call does not affect the presentation of the table.

The `TableModel` implementation used by `WTHtmlTable` must implement the interface, `wt.templateutil.table.AddColumn`. The `TableModel` does not have the column of checkboxes as part of its model initially. However, if `presentCheckBox` is set to true, then a column representing the checkboxes must be added to the `TableModel`. Adding a column that represents the checkbox column to the `TableModel` allows the `getValueAt` method to make sense to the `TableModel` for the checkbox column.

## Multiselect Following a Generic Search

To perform a multiselect following a generic search, perform the following steps:

1. Place the Windchill script in the HTML page that will perform the desired navigation and initialize the `WTHtmlTable`. The HTML page might present the results of a local search. In this case, the script call must be to a method that performs the following actions:
  - a. Performs the navigation.
  - b. Instantiates a `WTHtmlTable` object.
  - c. Sets the call the `setPresentCheckBox( true)` on the `WTHtmlTable` object.
  - d. Initializes an `WTHtmlTable` object with the results of the navigation.
  - e. Registers the `WTHtmlTable` with the HTML table service.

This step results in the first column in the HTML table being checkboxes. By default, there is no header for the column. To set a localized header on the column, you can use the tableService action=setHeaderFromResource script call. The name/handle of the column is checkBox.

2. Ensure the script calls, and hence the results, are inside an HTML Form with the proper return URL to process the multiselect.
3. Implement a FormTaskDelegate to receive the HTTP Post with the list of selected objects and process the list.

Subclass the wt.templateutil.processor.FormTaskDelegate class and implement the processAction method. In the processAction method, read the formData and process it. Then set the state so the correct response page is generated. Next register your new subclass of FormTaskDelegate in the properties file so the URL for the action of the Form results in your new FormTaskDelegate being used.

## Multiselect on an Association Navigation

To perform a multiselect on an association navigation, perform the following steps:

1. Place the Windchill script in the HTML page that will perform the desired navigation. See the section on the AssociationListTableService for the syntax.
2. Ensure that the USECHECKBOXTABLE=true is passed in the tableService action=initAssociationTable call. This results in the first column in the HTML Table being checkboxes. By default, there is no header for the column. To set a localized header on the column, you can use the tableService action=setHeaderFromResource script call. The name/handle of the column is checkBox.
3. Ensure the script calls, and hence the results, are inside an HTML Form with the proper return URL to process the multiselect.
4. Implement a FormTaskDelegate to receive the HTTP Post with the list of selected objects and process the list. Subclass the wt.templateutil.processor.FormTaskDelegate class and implement the processAction method. In the processAction method, read the formData and process it. Then set the state so that the correct response page is generated. Next register your new subclass of FormTaskDelegate in a properties file so the URL for the action of the Form results in your new FormTaskDelegate being used.

The following is the correct entry in an HTML template to generate a list and present the list in a table where the first column is a list of checkboxes. The first column will have the title "Delete".

```
<Form name="multiSelect"
action="...wt.enterprise.URLProcessor/
processForm?action=processMultiSelect" method=POST>

<SCRIPT LANGUAGE=Windchill>
<!--
tableService action=initAssociationNavigation ROLE=uses
LINKCLASSNAME=wt.part.WTPartUsageLink OTHERSIDECLASS=
wt.part.WTPartMaster
tableService action=initAssociationTable othersideattributes=name,
number,type linkattributes=modifyTimestamp,quantity
USECHECKBOXTABLE=true
tableService action=setHeaderFromResource COLUMNNAME = checkBox
RESOURCEBUNDLE=wt.clients.change2.Change2RB
RESOURCEKEY=REMOVE
tableService action=show
-->
</SCRIPT>

</FORM>
```

The following is a skeleton implementation of the code that will process the information sent back from the table generated by the preceding steps. The method, processSelections, is responsible for processing the resulting list of objects. The entry will be part of defining how the response page is generated.

```
public void processAction( ContentHTTPStream contentStream )
    throws Exception
{
    try
    {
        String selectionReference = null;

        String token = CheckBoxCellComponent.CONTEXT_SEPARATOR;
        Properties formData = getFormData();
        String keyName = null;

        Vector selections = new Vector();

        for ( Enumeration enum = formData.keys();
            enum.hasMoreElements() ; )
        {
            keyName = (String)enum.nextElement();
            if (keyName.startsWith(CheckBoxCellComponent.BEGIN_CHECKBOX_NAME))
            {
                String id = formData.getProperty( keyName );

                StringTokenizer idTokenizer = new StringTokenizer(id,
                    token);
                String skip_token = idTokenizer.nextToken();
                String oid = idTokenizer.nextToken();
```

```

        ReferenceFactory rf = new ReferenceFactory();
        WTReference ref = rf.getReference(oid);

        selections.addElement(ref.getObject());
    }
}

processSelections( selections, selectionReference,
    getContextObj() );

setContextAction( "&lt;Successful Response Action>" );

return;
}
catch (WTEException wte)
{
    wte.printStackTrace();
    addToResponseFooters((LocalizableMessage)wte);
    setContextAction( "CreateAssociationFailed" );
}
}

```

The FormTaskDelegate subclass also must be registered in one of the properties files so the FormTaskDelegateFactory is able to find it. Based on the URL in the preceding Form, the FormTaskDelegateFactory will look for the following entry:

```

wt.services/svc/default/
    wt.templateutil.processor.FormTaskDelegate/
        processMultiSelect/java.lang.Object/0=
        <The FormTaskDelegate for processing the MultiSelect>/
        duplicate

```

## Multiselect on the Contents of a Business Object

The multiselect for the contents of a business object is slightly different. First, you use the ListContentTableService instead of the AssociationListTableService. Second, because the ListContentTableService allows individual control over the ApplicationData table and the URLData table, you can have both tables with checkboxes or only one of the tables with checkboxes. The steps listed below apply to both tables.

1. Place the Windchill script in the HTML page that will use the ListContentTableService. See the section on the ListContentTableService for the syntax.

2. Ensure that the USECHECKBOXTABLE=true is passed in the tableService action= initURLDataTable (for URLData) and USECHECKBOXTABLE=true is passed in the tableService action= initApplicationDataTable (for ApplicationData). This results in the first column in the HTML table being checkboxes. There is no header for the column. To set a localized header on the column, you can use the tableService action=setHeaderFromResource script call. The name/handle of the column is checkBox. You must do this for each table separately. This also allows the option of different column headers if you choose.
3. Ensure that all tables that have checkboxes are inside an HTML Form with the proper URL to process the multiselect.
4. Implement a FormTaskDelegate to receive the HTTP Post with the list of selected objects and process the list.

Subclass the wt.templateutil.processor.FormTaskDelegate class and implement the processAction method. In the processAction method, read the formData and process it. Then set the State so that the correct response page is generated. Then register your new subclass of FormTaskDelegate so the URL for the action of the Form results in your new FormTaskDelegate being used.

The following is the correct entry in an HTML template to generate a list and present the list in a table where the first column is a list of checkboxes. The first column will have the title "Delete".

```
<Form name="multiSelect"
action="...wt.enterprise.URLProcessor/
  processForm?action=processMultiSelect" method=POST>

<SCRIPT LANGUAGE=Windchill>
<--
tableService action=initializeContents
tableService action=initURLDataTable USECHECKBOXTABLE=true\
  URLDATAATTRIBUTES=urlLocation,description
tableService action=show
-->
</SCRIPT>

<SCRIPT LANGUAGE=Windchill>
<!--
tableService action=INITAPPLICATIONDATATABLE USECHECKBOXTABLE=true
  APPLICATIONDATAATTRIBUTES=fileName,format,fileSize,
  modifyTimestamp,createdBy
tableService action=show
-->

</FORM>
```

## Adding a Column of Checkboxes to Any Subclass of HTMLTable

You might want to use a subclass of HTMLTable that does not implement the CheckBoxColumnAble interface. You can still add a column of checkboxes and enable multiselecting. The only requirement is that the TableModel you are using implements the AddColumn interface.

The subclass of HTMLTableColumn, `wt.templateutil.table.CheckBoxTableColumn`, is used by `WTHtmlTable` to generate the column of checkboxes. If you want to arbitrarily add a column of checkboxes to an HTMLTable through Windchill script, make the following call:

```
tableService action=addColumn name=checkbox COLUMNCLASS=  
wt.templateutil.table.CheckBoxTableColumn
```

# Changing the Presentation of the Attributes in the HTML Client

This section describes the process for modifying how attribute names and attribute values are presented in the HTML client through the Windchill script calls, `objectPropertyName`, and `objectPropertyValue`.

## Overview

Currently, if a name/value pair display of an attribute on the current Context Object of the page is desired, the following lines of HTML code are used:

```
<TR>
<TD ALIGN=RIGHT>
<B>
<SCRIPT LANGUAGE=Windchill>
<!--
objectPropertyName propertyName=lifeCycleState
-->
</SCRIPT>:
</B>
</TD>

<TD>
<SCRIPT LANGUAGE=Windchill>
<!--
objectPropertyValue propertyName=lifeCycleState
-->
</SCRIPT>
</TD>
</TR>
```

The preceding lines have the following default responses:

### **objectPropertyName**

The display name of the modeled attribute is retrieved from a resource bundle using Windchill introspection.

### **objectPropertyValue**

The value of the attribute is retrieved from the current context object using reflection to find a getter method with the following signature:

```
public <Return Type> getLifeCycleState( void )
```

This process for displaying the name/value pair can be overridden in two ways:

- Go into the existing source code and make changes or override certain methods in your template processor (this method is not preferred).
- Use the HTML components and create a component to handle generation of the HTML for you (this method is preferred).

The remainder of this section discusses the steps involved in creating an HTML component and using it to override the default name/value display generation.

## How the Display is Generated by Default

### Displaying the Name of the Attribute

For the following Windchill script call:

```
<SCRIPT LANGUAGE=Windchill>
<!--
objectPropertyName propertyName=lifeCycleState
-->
</SCRIPT>:
```

the following steps are enacted:

1. The name of the attribute is read from the "propertyName" of the script call.
2. The current context object is retrieved.
3. The HTMLComponentFactory is called to retrieve the HTML component to generate the display.

The HTMLComponentFactory returns HTML components based on application context service properties with the service name `wt.templateutil.components.HTMLComponents`. The default entry in the `htmlcomponent.properties` that the HTMLComponentFactory looks for is of the following form:

```
wt.services/rsc/default/
wt.templateutil.components.HTMLComponent/NAME/
<Class of Attribute>/0=<HTML Component to use>
```

In this case, the *<Class of Attribute>* refers to the return type of the getter method for the attribute. The current default entry is as follows:

```
wt.services/rsc/default
/wt.templateutil.components.HTMLComponent/NAME/
java.lang.Object/0= wt.templateutil.components.NameComponent
```

This means that the class `wt.templateutil.components.NameComponent` is currently used to generate the name for all classes.

4. The `init` method is invoked on the HTML component.
5. The `show` method is invoked on the HTML component.

## Displaying the Value of the Attribute

For the following Windchill script call:

```
<SCRIPT LANGUAGE=Windchill>
<!--
objectPropertyValue propertyName=lifeCycleState
-->
</SCRIPT>
```

the following steps are enacted:

1. The name of the attribute is read from the properties of the script call.
2. The current context object is retrieved.
3. The HTMLComponentFactory is called to retrieve the HTML component to generate the display.

The HTMLComponentFactory returns HTML components based on application context service properties with the service name `wt.templateutil.components.HTMLComponents`. The default entry in the `htmlcomponent.properties` that the HTMLComponentFactory looks for has the following form:

```
wt.services/rsc/default/wt.templateutil.components.HTMLComponent/
null/<Class of Attribute>/0=<HTML Component to use >
```

In this case, the *Class of Attribute* refers to the return type of the getter method for the attribute.

One current default entry is as follows:

```
wt.services/rsc/default/wt.templateutil.components.HTMLComponent/
null/java.lang.Object/0
=wt.templateutil.components.DefaultBusinessComponent
```

4. The `init` method is invoked on the HTML component.
5. The `show` method is invoked on the HTML component.

## Overriding the Display with an HTML Component

Overriding the name display and overriding the value display use almost the same mechanism. Therefore, the steps to perform them are almost identical. Only the *<Class of Attribute>* value differs, as explained in the last step.

1. Modify the Windchill script call by adding `serviceSelector=<Service_Name>` to your Windchill script call:

```
<SCRIPT LANGUAGE=Windchill>
<!--
objectPropertyName propertyName=lifeCycleState
serviceSelector=<Service_Name>
-->
</SCRIPT>:
```

where `<Service_Name>` is a name of your choice.

2. Create a subclass of `wt.templateutil.components.HTMLComponent` to generate the desired HTML. The preceding section on HTML components, HTMLTable, and HTML table service discusses this. The following example also goes through these details.
3. Add an application context service property with the following form:

```
wt.services/rsc/default/  
wt.templateutil.components.HTMLComponent/<Service_Name>/  
<Class of Attribute>/0=<New HTML Component>
```

When overriding the name display, the `<Class of Attribute>` value refers to the return type of the getter method. If you want to write some generically available component that is based solely on the `<Service_Name>` and does not select based on class, you could set the `<Class of Attribute>` equal to `java.lang.Object`.

When overriding the value display, the `<Class of Attribute>` value refers to the return type of the getter method.

## Adding a Link to an Attribute Value

This example describes how to override the presentation of the value of the `lifeCycleState` attribute of a `wt.part.WTPart` in the object properties page. This has two implications:

- The context object will be a `wt.part.WTPart` class.
- The HTML template to edit will be `/codebase/templates/ObjectProperties/WTPart.html`.

To perform this example, follows these steps:

1. Modify the Windchill script call by adding `serviceSelector=MyLink` to your Windchill script call as follows:

```
<SCRIPT LANGUAGE=Windchillgt;  
<!--  
objectPropertyValue propertyName=lifeCycleState  
  serviceSelector=MyLink  
-->  
</SCRIPT>:
```

2. Create a subclass of `wt.templateutil.components.HTMLComponent` to generate the desired HTML for you.

Because a link is being added, the `wt.templateutil.components.HTMLLinkComponent` class must be subclassed. This is done because the `HTMLLinkComponent` is the base component that knows how to generate HTML that looks like the following:

```
<a "The rest of the HTML from startElement method">"HTML from  
showSubComponents method"</a>
```

The primary methods to override are the `init` method and the `startElement` method, and possibly the `showSubComponents`. The `init` method is overridden so that a subcomponent can be added. Then, when the `showSubComponents` method is called, the added subcomponent will generate HTML. The subcomponent will generate the correct text which will appear as the hyperlinked text in the link. The `startElement` will be overridden to generate the correct link by setting the value of the `href` attribute. (The source file is attached at the end of this section.)

After creating the source for `wt.templateutil.components.MyLinkComponent`, compile it.

3. Add an application context service property (all on one line) with the following form:

```
wt.services/rsc/default/  
wt.templateutil.components.HTMLComponent/  
MyLink/wt.state.State/  
0=wt.templateutil.components.MyLinkComponent
```

Save the file.

In this example, the `<Class of Attribute>` value refers to the class of the attribute, not the class of the object that has the attribute. The class of the `lifeCycleState` attribute is `wt.state.State`. This value is used, not the class of the business object with the attribute.

Restart the method server and go to a properties page for a `wt.part.WTPart`. You should see the attribute `State` with a hyperlink.

## The Source Code for the MyLink Example

```
package wt.templateutil.components;  
  
import java.lang.String;  
import wt.templateutil.components.HTMLLinkComponent;  
  
import java.io.OutputStream;  
  
import java.net.URL;  
import java.util.HashMap;  
import java.util.Locale;  
import java.util.Vector;  
import java.util.Properties;
```

```

import wt.fc.Persistable;

import wt.htmlutil.HtmlUtil;

import wt.httpgw.GatewayServletHelper;

import wt.templateutil.processor.PageContext;
import wt.templateutil.processor.HTTPState;

import wt.templateutil.table.WTAttribute;

import wt.util.WTException;
import wt.util.WTMessage;
import wt.util.WTProperties;

/**
 *
 * This class is a demonstration of how to implement adding a link to an
 * attribute that is being displayed in an Object Properties page. This
 * extends the wt.templateutil.components.HTMLLinkComponent class because
 * the main purpose of this class is to wrap the display value in a link.
 *
 * @version 1.0
 */
public class MyLinkComponent extends HTMLLinkComponent
{
    private static final String RESOURCE =
        "wt.templateutil.components.componentsResource";
    private static final String CLASSNAME = MyLinkComponent.class.getName();
    private static final String versionID = "$Header: $";

    private static boolean VERBOSE = false;

    static
    {
        try
        {
            WTProperties properties = WTProperties.getLocalProperties();

            // Standard trick to read entries from wt.properties to decide
            // if the
            // System.err statements are printed.
            VERBOSE = properties.getProperty
("wt.templateutil.components.verbose", false);
            VERBOSE = properties.getProperty
("wt.templateutil.components.MyLinkComponent.verbose",
VERBOSE);

        }
        catch (Throwable t)
        {
            t.printStackTrace(System.err);
            throw new ExceptionInInitializerError(t);
        }
    }
}

```

```

}

/**
 * This method overrides the default implementation to add a subcomponent
 * to this component.
 * The subcomponent will be responsible for presenting the text to be
 * displayed in the
 * link in the Object Properties page.
 *
 * @param selector The String used by the HTMLComponentFactory
 * @param value The object used by the HTMLComponentFactory
 * @param componentFactory The current instance of the
 * HTMLComponentFactory, if available
 * @param mode Can be used to indicate is the current mode is
 * View or Update
 * @param props The Properties object passed in by the Windchill
 * Script call.
 */
public void init( String selector, Object value, HTMLComponentFactory
    componentFactory, String mode, Properties props )
{
    if (VERBOSE)
    {
        System.err.println("\n\n MyLinkComponent : in init \n\n");
    }

    // Call to super.init is not required, but HIGHLY recommended.
    // Default clean up
    // is performed as well as reading default display values from
    // wt.properties and
    // using any entries in the props object that was passed in to
    // initialize values
    super.init( selector, value, componentFactory, mode, props );

    // Create the default HTML Component that generates the display of an
    // attribute. Then add the attribute to this HTML Component. The result
    // will be that when this HTML component's showSubComponents method is
    // invoked, this added component's show method will be called. This
    // will result in the text to be displayed with the link.

    DefaultBusinessComponent subComponent = new DefaultBusinessComponent();
    subComponent.init( selector, value, componentFactory, mode, props );
    getHtmlComponents().addElement( subComponent );
}

/**
 * This method is responsible for generating the starting tag for an
 * HTML Element.
 * In the case of a link, the starting tag would begin as,
 * <a href=...>. This
 * method is responsible for generating this starting tag.
 *
 * @param value The object of current interest
 * @param formData The formData from the HTTP Request
 * @param os The OutputStream of the HTTP Response
 * @param locale The current Locale of the HTML Template

```

```

* @return String The Starting tag of an HTML link Element
**/
public String startComponent( Object obj, Properties formData,
    OutputStream os, Locale locale )
{
    if (VERBOSE)
    {
        System.err.println("\n\n MyLinkComponent : in startComponent \n\n");
    }

    // Using a StringBuffer for efficiency
    StringBuffer startStr = new StringBuffer(50);
    if (VERBOSE)
    {
        System.err.println( "\n DefaultLinkComponent - startComponent :
            getTag() = " + getTag());
    }
    startStr.append( "&lt;" );
    startStr.append( getTag() );
    startStr.append( " href=\"\" );

    // This is the standard manner to generate a proper URL in Windchill.

    // The following code will retrieve the Context Object of the
    // current page, IF
    // the call is coming from objectPropertyValue. There are other
    // cases will this
    // will work also. But it will not always work.
    String oidString = "";
    Persistable contextObject = null;
    if ( obj instanceof wt.templateutil.table.WTAttribute )
    {
        contextObject = (Persistable)
            ((wt.templateutil.table.WTAttribute)obj).getSource();
    }
    else
    {
        contextObject = (Persistable)obj;
    }

    wt.fc.ReferenceFactory factory = new wt.fc.ReferenceFactory();
    try
    {
        oidString = factory.getReferenceString( contextObject );
    }
    catch (WTEException wte)
    {
        if (VERBOSE)
        {
            wte.printStackTrace();
        }
    }

    // Set the name/values pairs to appear on the Query String on the URL

```

```

HashMap queryString = new HashMap();

queryString.put( "oid" , oidString );
queryString.put( "action" , "ObjProps" );

try
{
    // Generate the URL and add to the return String
    startStr.append(
GatewayServletHelper.buildAuthenticatedHRef(this.getState().getURLFactory(),"wt.ente
prise.URLProcessor","URLTemplateAction", query String));
    }
    catch (WTEException wte)
    {
        if (VERBOSE)
        {
            wte.printStackTrace();
        }
    }

    // Close the Start tag
    startStr.append( "\">" );

    // Return the value
    return startStr.toString();
}
}

```

# Setting Color and Style Attributes in HTML and JSP Clients

This section describes how the color and other style aspects of Windchill PDM HTML pages may be modified. The look of ProjectLink and PDMLink JSP pages is derived primarily from the stylesheet, `<WT.HOME>/codebase/netmarkets/css/nmstyles.css`. That stylesheet also controls the style of the page header, page footer and tabs of ProjectLink template processing pages. See the stylesheets for more information.

Color, font, and other style attributes of Windchill PDM page elements are set in various ways, including the following:

- Referencing the Windchill stylesheet

The Windchill stylesheet is used in relatively few clients at this time. However, its use should increase in future releases and it is recommended for new clients.

- Referencing color and font properties in `wt.properties`

Currently, most Foundation clients use this method. Style attributes other than color and font face are generally hard-coded.

- Hard-coded values

The use of hard-coded values for color and fonts is being phased out and is not recommended.

- Feature-specific methods

## Windchill Stylesheet

The color, font, and other style attributes of Windchill PDM page elements can be set by referencing the style classes in the Windchill stylesheet, `WTDefault.css`, which is located in `Windchill/codebase`. For further information on the application of style classes, see the section on Customizing Colors in HTML and JSP Clients in chapter 4.

## Referencing the Windchill Stylesheet from an HTML Template

If an HTML template contains a `createBase` Windchill script call, the stylesheet can be referenced as follows:

```
<SCRIPT LANGUAGE=Windchill>
wt.htmlutil.HtmlUtil createBase
</SCRIPT>

<HEAD>
.
.
.
  <LINK rel="stylesheet" type="text/css" href="WTDefault.css">
.
.
```

```
.  
</HEAD>
```

If the HTML template does not have a `createBase` call, the `href` attribute should contain the stylesheet path relative to the current page, for example:

```
../../WTDefault.css
```

## Referencing the Windchill Stylesheet from a JSP Page

On a JSP page, use the `URLFactory` to generate the stylesheet link. For example:

```
<jsp:useBean id="url_factory" class="wt.httpgw.URLFactory"  
  scope="request" >  
<%  
    url_factory.setRequestURL(request.getScheme(),  
      request.getHeader("HOST"), request.getRequestURI());  
%>  
.br/>.br/><HEAD>  
.br/><link rel="stylesheet" type="text/css" href=  
  "<%=url_factory.getHREF("WTDefault.css")%>">  
.br/></HEAD>
```

## Color and Font Properties in `wt.properties`

The file `Windchill/codebase/wt.properties` contains a number of color properties used for setting colors in HTML templates. Their names begin with `"wt.html.color..."` and reference the same style classes and colors as the stylesheet. These properties are described in more detail in the section on `Customizing Colors in HTML and JSP Clients` in chapter 4.

The `wt.properties` file also has a property named `wt.html.font-family` that gives the font faces typically used in Windchill pages, in order of preference.

If you want to modify any of these properties, you should not modify the `wt.properties` file directly as your modifications may be overwritten if Windchill is reinstalled or upgraded. You should instead, create a new entry for the property. This can be done in various ways. See the section, `The Windchill Development Environment`, in the `Windchill Application Developer's Guide` for more information.

## Referencing Color and Font Properties from an HTML Template

The following Windchill script method in `BasicTemplateProcessor` can be used to generate a hexadecimal color number on an HTML page based on a lookup of a color property in `wt.properties`:

```
public void getWCColor(Properties parameters, Locale locale,
                      OutputStream os) throws WTEException
```

This script call has a required parameter "style class" to indicate the style class name of the desired color and an optional parameter "quotes" to indicate whether single, double, or no quotes are desired around the color hexadecimal number string written to the page.

Following is an example of usage:

```
<BODY BGCOLOR=<SCRIPT LANGUAGE=Windchill>getWCColor styleClass=
  bg-body</SCRIPT> TEXT=<SCRIPT LANGUAGE=Windchill>getWCColor
  styleClass=f-body</SCRIPT> >
```

This method can also be called recursively to override default colors used by the Windchill HTML Table Service. For example:

```
<SCRIPT LANGUAGE=Windchill>
<!--
showObjectSearch
  tableService action=setHeaderAttributes COLUMNNUMBER=
  ALL th.ALIGN=left font.size=2 recurse.th.bgcolor=
  "wt.enterprise.BasicTemplateProcessor getWCColor styleClass=
  bg-hilitel"
tableService action=show
-->
</SCRIPT>
```

In a similar manner, the font face attribute can be generated with the following script call:

```
public void getWCFontFamily(Properties parameters, Locale locale,
                             OutputStream os)
```

This does not have a style class parameter but does have an optional "quotes" parameter.

For example:

```
<HEAD>
<STYLE TYPE="text/css">
<!--
  P, FONT, B {font-family: <SCRIPT LANGUAGE=
  Windchill>getWCFontFamily quotes=none</SCRIPT>}
-->
</STYLE>
</HEAD>
```

Additional examples are shown in the example template named `Example1.doc` in the `Windchill/src/customization/HTMLClient` directory.

## Referencing Color and Font Properties from an HTML Template Processor or JSP Page

The following method in `wt/enterprise/BasicTemplateProcessor` can be used within HTML template processors and JSP pages to retrieve color property values from `wt.properties`:

```
public static String getWCColor(String styleClass)
```

This returns a hexadecimal number string in the form `#NNNNNN`.

Following is example usage in a template processor:

```
PrintWriter out;  
.  
.  
.  
String headercolor = getWCColor("t1-bg-col-head");  
String headerFontColor = getWCColor("t1-f-col-head");  
out.println("<TH BGCOLOR=" + headercolor + " ALIGN=LEFT  
    <FONT COLOR=\"" + headerFontColor + "\"><B>");
```

The corresponding method to get the font face property value is:

```
public static String getWCFontFamily()
```

Following is example usage in a template processor:

```
tableFontFace = " FACE=\"" + getWCFontFamily() + "\"";  
out.println("<FONT " + tableHeaderFontColor + tableFontFace + "  
    SIZE=\"" + tableHeaderFontSize + "\">\n");
```

# HTML Component Implementation

This section describes how the HTML component functionality was implemented in Windchill.

## The General Structure

The Windchill approach to presenting the structure of HTML UI components is based on the Java implementation of user interface components. This implementation contains the following:

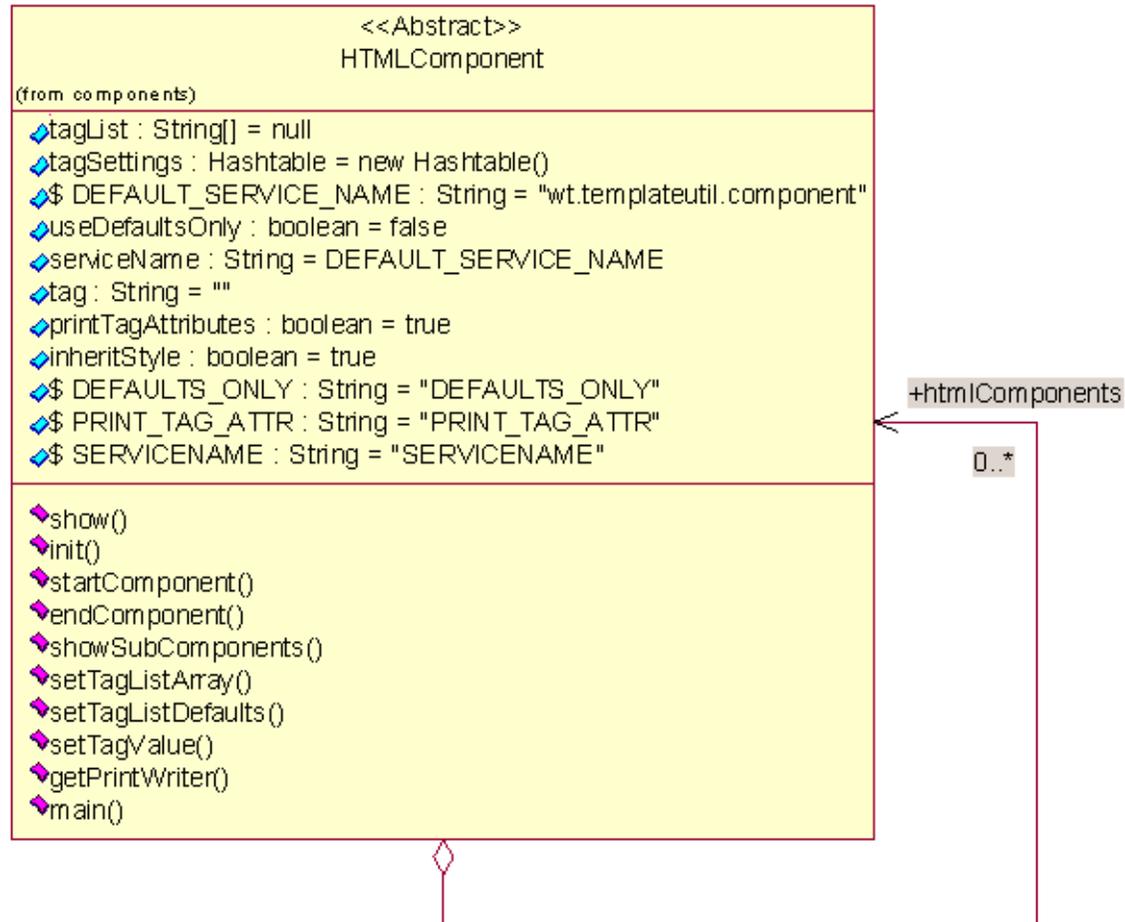
- A base interface that defines the core functionality of a component and how the components work together.
- A set of peer classes that know how to present the view of the basic HTML UI components in the client environment, in this case, an HTML browser.
- A set of custom classes that extend the base classes and interfaces, and use the peer classes. These custom classes are usually, but not always, composites of several peer classes, and have some business logic.

The following sections describe the implementation of this structure.

## The HTMLComponent Class

### Overview

The HTMLComponent class is the interface that all HTML UI components must implement.



This class represents the calls to be made to initialize and present the UI component, and it provides the structure for embedding UI components within other UI components. It also provides some default implementations for initializing and printing the HTML component.

## The init() Method

Following is the full signature of the init method:

```
public void init( String selector, Object value,  
HTMLComponentFactory componentFactory, String mode,  
Properties props)
```

The init method is responsible for the following actions:

- Reading the default settings from wt.properties based on the current service name.
- Overriding the default settings in wt.properties with entries passed in the Properties object.
- Loading any subcomponents.
- Setting any state information.

## The show() Method

The show method call causes the component to be rendered, by printing HTML, to the browser. The implementation of the show() method should account for any aggregated instances of HTMLComponent. The show() call should be implemented like the HTML elements: starting tag, text, and ending tag.

Following is the show() method as it appears in the HTMLComponent class:

```
public String show( Object value, Properties formData,  
    OutputStream os, Locale locale ) {  
  
    StringBuffer componentStr = new StringBuffer(200);  
    componentStr.append( startComponent( value, formData,  
        os, locale ) );  
    componentStr.append( showSubComponents(value, formData,  
        os, locale) );  
    componentStr.append( endComponent( value, os, locale) );  
    return componentStr.toString();  
}
```

The following methods are called in the show() method:

```
public String startComponent( Object value,  
Properties formData, OutputStream os, Locale locale)
```

This method prints the starting tag of an HTML element. The most important responsibility of this method is to print the attributes for the HTML element.

```
public String showSubComponents( Object value,  
Properties formData, OutputStream os, Locale locale)
```

This method is responsible for printing the text that appears between the start and end tags of the HTML component. The default implementation of this method in HTMLComponent is to enumerate over the subcomponents and call the show method on each subcomponent.

```
public String endComponent( Object value, OutputStream os,  
Locale locale)
```

This method is responsible for returning the end tag of the HTML component.

## Efficiency Issues and the init Method

Efficiency is an important consideration in rendering HTML pages, even more so than in Java clients. In the process of generating local search results with 50 returns, each row having 5 columns, at least 250 objects are created if you implement the HTMLComponents in the obvious way.

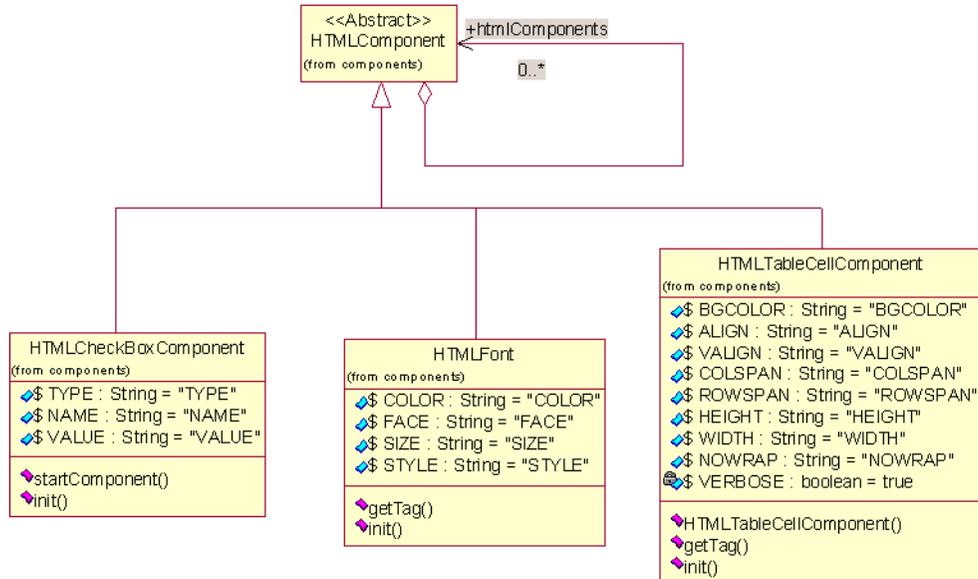
To implement these calls as efficiently as possible, object reuse should be done whenever possible. In whatever service is provided to locate and initialize the HTMLComponents, there should be an object caching service. One result of this implementation is to allow the use of cached instances of HTMLComponents. A second result is that the HTMLComponents must be able to be refreshed after each use. By nature, all HTML components are stateless. They are driven by the current value with which they are presented. To ensure that the state of the component is fresh for the next rendering, the init call allows resetting of the HTML component's state.

## HTML Peer Components

HTML peer components represent the base implementation of a specific HTML component. The tag sets in HTML represent the various HTML UI components including, but not limited to, the following:

- Font
- A (links)
- Table
- TD (table cell)
- Select

There are convenience classes to be used to render the HTML specific views. The developer can call on these classes and set values without knowing specifically how to set the HTML view. These classes present an API to allow custom UI components (described later in this section) to present their information. For example, most developers do not really care how a drop-down list is generated. Ideally, they want an API that allows them to pass in a two-dimensional array to a component and have the component do the rest. Furthermore, these base classes should know how to read in the default settings for presenting their view, that is, default colors, default fonts, and so on.



## Initializing Default Values

The HTMLComponents allow the setting of system-wide default values for presentation characteristics using the `wt.properties` file and a naming convention to indicate the components and the values, such as the following:

```
wt.templateutil.TableCell.bgcolor=red
```

The option exists to override the default values, either through the script calls or programmatically overriding the values.

## Reading Defaults from the `wt.properties` File

The default values are retrieved from the `wt.properties` reading entries with the following format:

```
<Service Name>.<HTML Element>.<Attributes>=<Value>
```

Following are examples of setting the background color for an HTML table cell:

```
wt.templateutil.component.TD.BGCOLOR=
 $(wt.html.color.t1-bg-evenrow)
wt.templateutil.component.TH.BGCOLOR=#DDDDDD
```

This entry assumes that the `serviceName` field on the HTML component is set to `"wt.templateutil.component"`. This is the default value for all HTML components. By resetting the value of the `serviceName`, you can reference different default settings in the `wt.properties` files.

## Overriding Values Using the `init` Method

The `init` method provides the option of setting presentation values by passing in overriding values in the `Properties` object with name and value pairs. The name must be a static Strings token defined by the HTML component and the value must be the setting for that token. For example, the following code initializes a `Property` object to have the background color token, `COLOR`, set to blue:

```
Properties myProps = new Properties();
myProps.put( HTMLFont.FONT_COLOR,
    BasicTemplateProcessor.getWCColor("f-hilite"));

HTMLFont myFont = new HTMLFont();
myFont.init( "tableCell", value, componentFactory, mode,
    myProps);
```

This technique was used on the premise that most of the time, if a default value is to be overridden, it will be from a script call in an HTML template. The name and values pairs of the Windchill script calls are passed in a `Properties` object. By passing in this `properties` object, override values can be passed in for presentation.

## Setting Presentation Values Programmatically

You can also change presentation values programmatically. Suppose that HTML component A contains HTML component B, and A wants to set a value on B. To accomplish this, you can use the technique just described (override the value using the `init` method). However, to do so, you must catch the `Properties` object being passed to the `init` call and set the values on it. If you want HTML component A to be able to set a value on B directly, use the following `setTagValue` method:

```
public void setTagValue( String tagID, String tagValue)
```

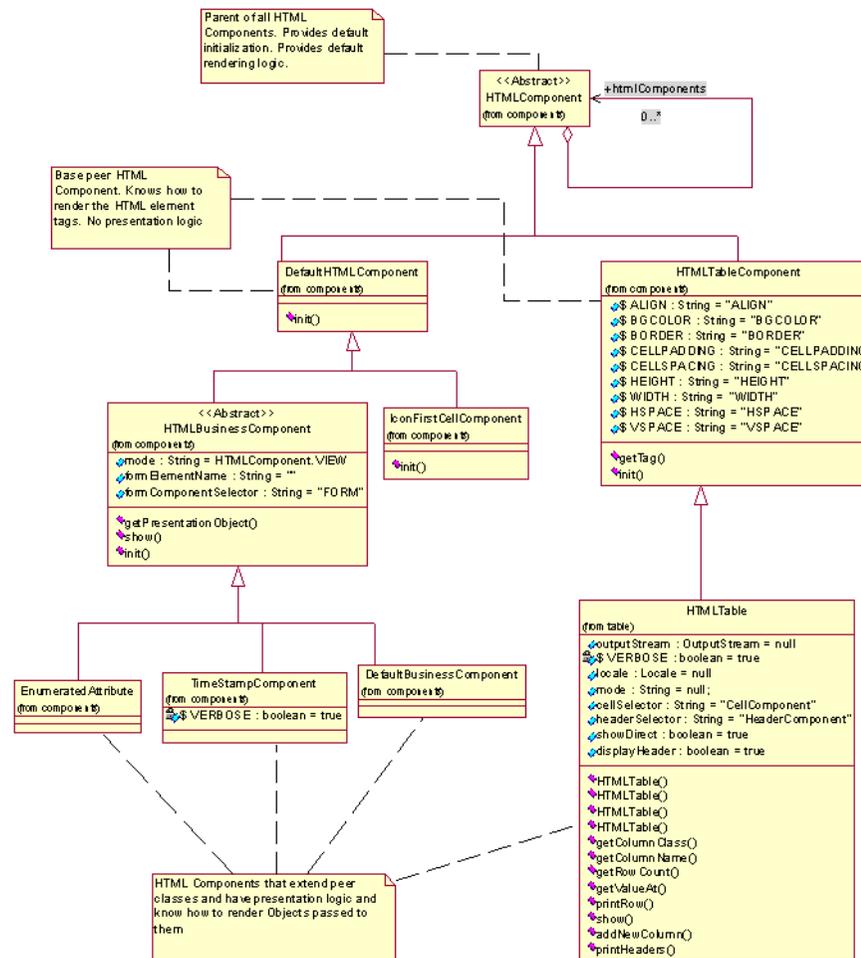
The arguments in this method are the same name and value pairs that you would pass in the `Properties` object in the `init` method. For example, the following code has the same effect as the code shown in the preceding section:

```
HTMLFont myFont = new HTMLFont();
myFont.init( "tableCell", value, componentFactory, mode,
    myProps);
myFont.setTagValue(HTMLFont.FONT_COLOR,
    BasicTemplateProcessor.getWCColor("f-hilite"));
```

**Note:** Calling the `setTagValue` method before calling the `init` method is not recommended. The `init` method overrides any preset values.

## Custom Components

These classes present the information of business classes and also present composite components, for example, IconFirstColumnComponent. This is where most development work of the developer is. You implement the HTMLComponent interface and create a component that knows how to present specific information for a business class using the default implementations of the parent classes.



## The Factory Service Implementation for HTML Components

The ApplicationContextChild factory method uses information from two keys:

- A String that represents the context for selecting the HTML component.
- A class that represents the class of the object to be displayed.

The ability to find the correct HTML component is similar to a Swing `CellRenderer` because it is based on the class of the object to display. The difference is that, where you would use different `CellRenderers` at different times, using this technique you use a different context `String`.

An additional feature of the `ApplicationContextChild`-based Factory is that inheritance is used to locate the HTML component. If you specify a display class, `A2`, that is a subclass of `A1`, and there is no HTML component for `A2`, then the factory tries to find an HTML component for a display class of type `A2`.

# HTML Table Generation and Table Service Implementation

This section describes how the HTML table generation and table service functionality was implemented in Windchill. To understand this section, you should be familiar with the Swing JTable and the Swing TableModel, in particular.

## HTMLTable

This is the controller in the Model-View-Control (MVC) paradigm. It is responsible for communicating between the TableModel, the data, and the HTML View, the generated page. The HTMLTable acts as a proxy for the TableModel and the HTMLTableColumnModel. This can be seen in the UML model by looking at how the HTMLTable implements all of the methods in both the TableModel and the HTMLTableColumnModel. This is how the HTMLTable provides communication between the TableModel and the HTMLTableColumnModel. The implementation follows the implementation of the JTable.

One difference is that there is no event handling because there is no direct user interaction with the UI. However, after Servlets and JSP are used more commonly, the URLs might take on a much more event-based context because the Session context will allow the HTMLTable presented in a Web page to exist between HTTP requests.

The role of the HTMLTable, like the JTable, is to perform the following functions:

- Coordinate the communication of the TableModel and the HTMLTableColumnModel.
- Coordinate the rendering of the HTML Table by looping over the columns and rows of the HTMLTableColumnModel using the data in the TableModel.
- Provide a default mapping between the data classes in a given cell of the TableModel and the HTML component used to render the data in the table. In this case, the HTMLComponentFactory is used instead of a CellRenderer.

## Header Generation

The header generation is based in the interaction of the `HTMLTableColumn`, the `HTMLTable`, and the `TableModel`. There are two stages of Header Generation:

- Setting the header field based on the `TableModel`.

If a concrete subclass of `TableModel` also implements the `ColumnIdentifier` interface, then when a column is being added to the `HTMLTableColumnModel` during the initialization of the `HTMLTable`, the concrete subclass of `TableModel` is called to set the header field in the `HTMLTableColumn`. Later, when the header for the Table is generated, this value for the header is used. If the concrete subclass of `TableModel` does not implement the `ColumnIdentifier` interface, the header field of the `HTMLTableColumn` is not modified.

- Printing the headers as part of the show method call.

The process of printing the headers is fairly straightforward. For each column in the `HTMLTableColumnModel`, a call is made to get the current value of the header field on that column, if set, and the preferred `HTMLComponent` for presenting the header, if set.

If the header field was not set on the `HTMLTableColumn`, the `TableModel` is asked for the `columnName` of that column. The `String` returned is then used to display the column header.

If a preferred `HTML` component is not returned by the `HTMLTableColumn`, then the `HTMLComponentFactory` is used to locate an `HTML` component using the context `String` currently set in the field `headerSelector` in the `HTMLTable` and the class of the current header value (as just described) to locate the desired `HTML` component.

## Table Cell Generation

The table cell generation is based on the interaction between the `TableModel`, the `HTMLTable`, and the `HTMLTableColumn`, using the following process:

1. The current `HTMLTableColumn` is retrieved from the `HTMLTableColumnModel`.
2. The value for the current row and column is retrieved from the `TableModel`.
3. The current `HTMLTableColumn` is checked to see if there is a preferred `HTML` component to use to render the table cell.
4. If no preferred `HTML` component is returned, the `HTMLComponentFactory` is used to locate the desired `HTML` component. The context `String` used is the current value of the `cellSelector` field of the `HTMLTable` and the class used is the class of the value returned from the `TableModel`.
5. After the `HTML` component is returned, the show method is invoked on the `HTML` component.

## Configuration

Many fields can be set on the HTMLTable that affect the generation of its presentation. Following is a subset of the more important fields:

### **cellSelector**

Sets the context String for the HTMLComponentFactory when trying to find an HTML component for rendering a table cell.

### **headerSelector**

Sets the context String for the HTMLComponentFactory when trying to find an HTML component for rendering the header of a column in the table.

### **showDirect**

Toggles whether the HTMLTable behaves as an HTML component, returning the entire table in a single String, or the table behaves as a standalone table.

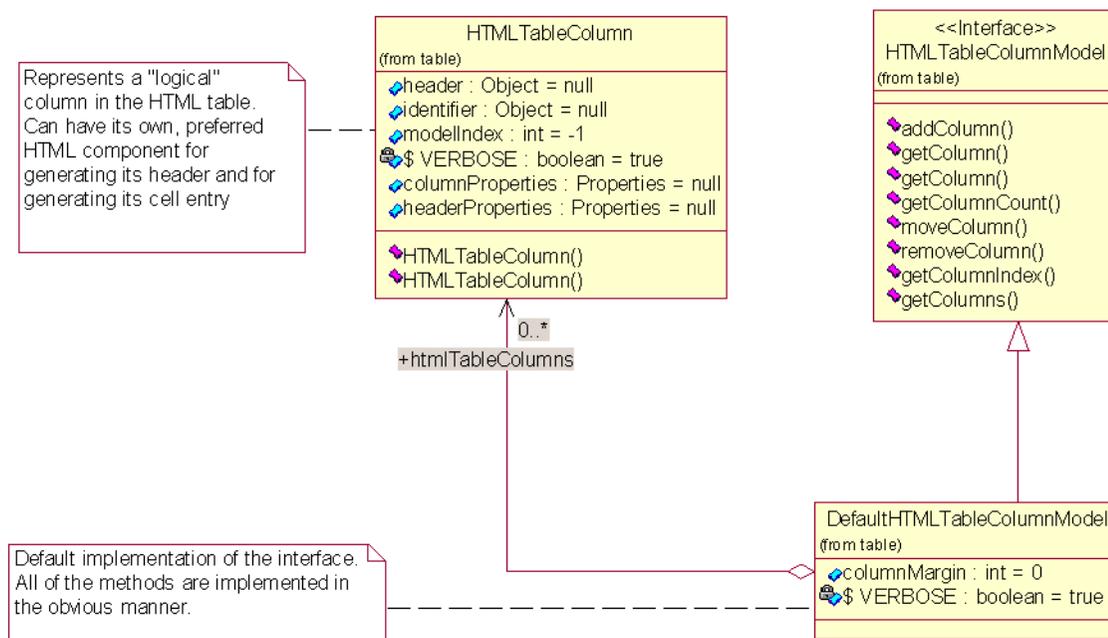
The HTMLTable is an HTML component. As such, its show method should return a String that is the entire table. For performance reasons, it is desirable to have the HTMLTable print directly to the output stream if the table is not part of another component; that is, if it is a stand-alone table. This field toggles how the HTMLTable behaves as an HTML component.



## HTMLTableColumnModel

A concrete instance of the HTMLTableColumnModel interface is responsible for determining the columns to present from the TableModel and the order in which to present them. As can be seen from looking at this interface, it is simply a group of methods that do this. The class DefaultHTMLTableColumnModel provides a default implementation of the HTMLTableColumnModel interface. This default implementation implements the methods in the obvious way.

One issue to consider in the future is to use a nonsynchronized collection class instead of the Vector instance that is currently being used. This would provide some better performance. Because the HTML table generation is serial, there is no issue of thread safety.



## HTMLTableColumn

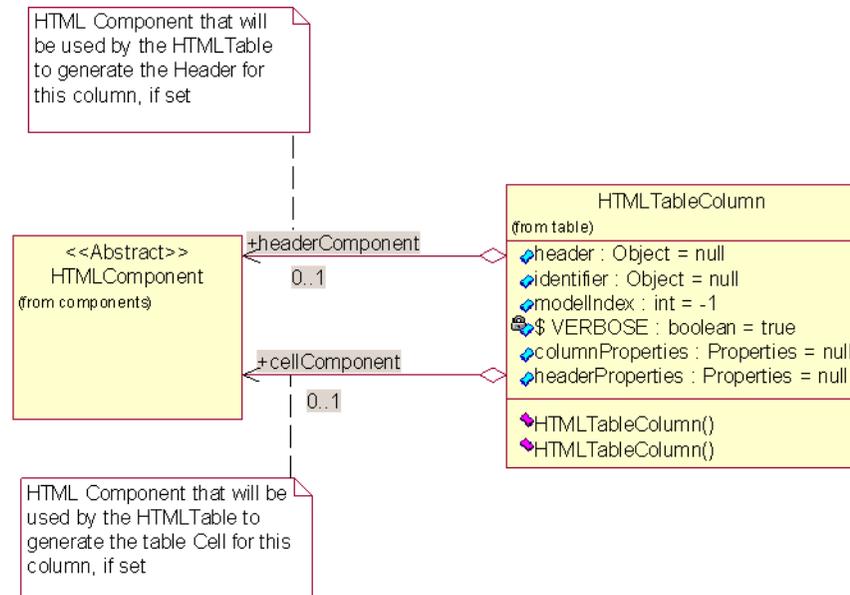
HTMLTableColumn is a base implementation of an object that represents a column in an HTML table. It maintains the following:

- (Optionally) A header object that is used to generate the title of the column in the HTML page.
- (Optionally) The HTMLComponent to present the title of the column based on the value of the header object.
- (Optionally) The HTMLComponent to present the values in the TableModel for this column.

- The index of the column in the TableModel that this column represents. This is used to retrieve the value of a cell in the TableModel from the correct Column.

The aggregated instances of the HTMLComponent are used to present the value of the object in the cell, if they are available. Otherwise, the HTMLComponentFactory in the HTMLTable retrieves an HTML component to present the value from the TableModel.

For each of the modeled attributes, there are getters and setters.



## HTMLComponentFactory

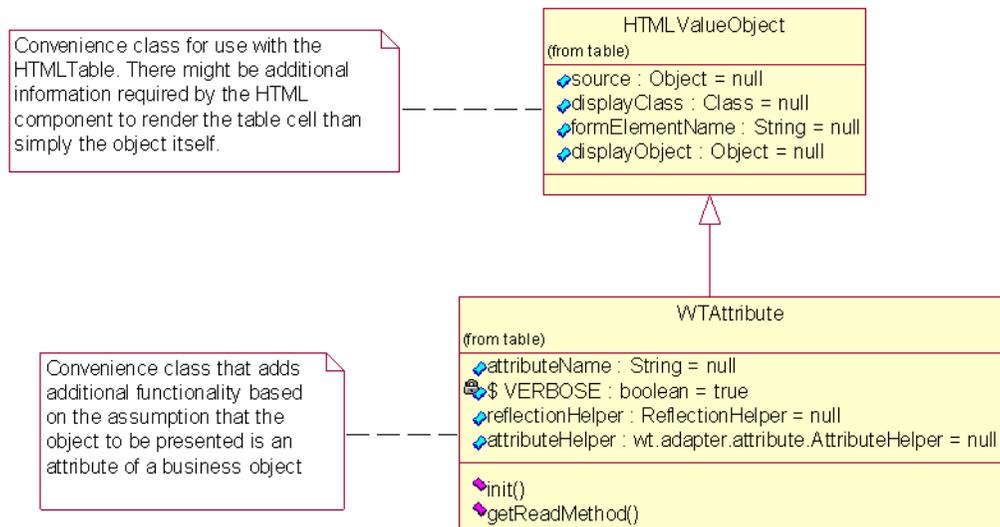
HTMLComponentFactory is the class that is responsible for deciding what HTMLComponent to use to generate the HTML, based on a value(object) from a cell in the TableModel. The selection of the HTMLComponentFactory is based on the class type of the object in the TableModel cell and a String that represents some context about how the HTML component is to be selected.

The HTMLComponent that is returned from the HTMLComponentFactory is then used to generate the HTML code to send back to the browser.

## HTMLValueObject and WTAttribute

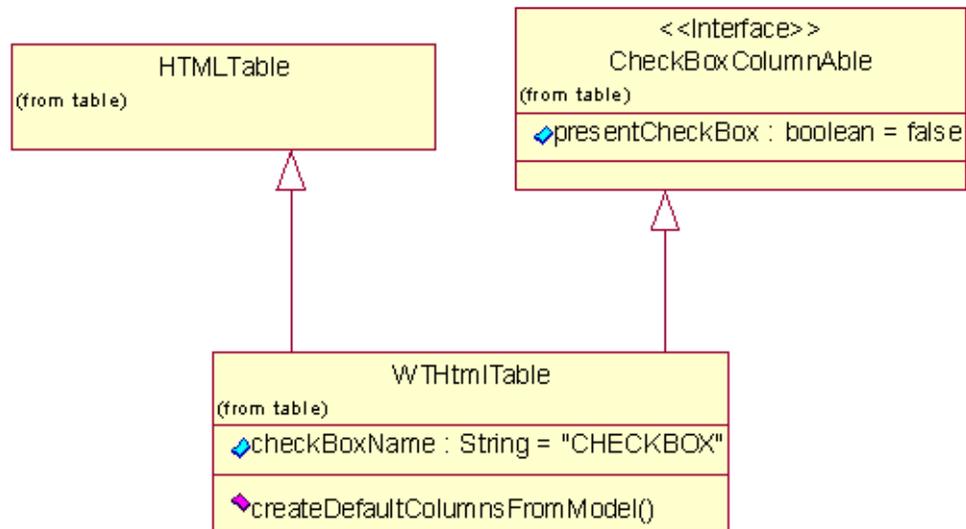
HTMLValueObject and WTAttribute are convenience classes to facilitate the break between the TableModel and generating the view using the HTMLTableColumnModel, HTMLTableColumn, and HTMLComponent triad. Usually you need only the object that is the value of the cell in the TableModel. However, with business class attributes, it is not that simple. To present the value of an attribute of a business class in HTML, you not only need the value of the attribute, but quite often the state of the parent object that attribute came from or some other information about the source of the attribute. To facilitate the break between the model and the view, the WTAttribute class was created to behave like the attribute but to carry the source of its value. This allows the passing of a single object that encapsulates the information to present the cell; that is, the value of the cell. Otherwise, additional communication would be needed between the TableModel and the view when an attribute of a business object is to be presented. This would not be consistent with the JTable type implementation and, more importantly, would result in a model that did not have a single mode of communication between the model and the view.

The HTMLValueObject is a parent class that does not provide any logic to work with business objects. It is an abstraction of the WTAttribute that provides a generic container to carry information between the TableModel and the HTML component that will render the cell in the HTML table.



## WTHtmlTable

The HTMLTable is the base implementation. An additional requirement is that a table be able to be generated with a checkbox on each row. This allows the HTML equivalent of the Multiselect. To support this requirement, the HTMLTable has been extended to allow the specification of having a column of checkboxes presented with the table. The checkbox column is treated like any other column in the sense that it has a header and should be able to be manipulated like the other columns using Windchill script calls (as described in the following section). The main point is that the same HTML table can be presented in a View Mode without the checkboxes or presented in an Update Mode with the checkboxes.



## The Table Service

As mentioned earlier, there is a general requirement to be able to affect the presentation of an HTML table from the Windchill script. This is to allow on-site customization of the presentation without having to go into the Java code.

Following are the primary requirements of the Windchill script calls:

- Add, Move, and Remove columns from the HTML table.
- Change the font and colors used in the HTML table.
  - Affect columns individually.
  - Affect headers and cells separately.
- Change border and width of the HTML table.
- Enable and disable checkbox.
- Enable and disable displaying the headers of the table.
- Select the HTMLComponent for a given column.

- Select the header from a localized resource bundle.

Following are the requirements on how the implementation of the Table Service was developed:

- The Table Service is implemented as a `TemplateProcessorService`. This requirement is based on the desire not to enlarge the `wt.enterprise.BasicTemplateProcessor` class and follow a more object-oriented design.
- The `TableService` is extensible.

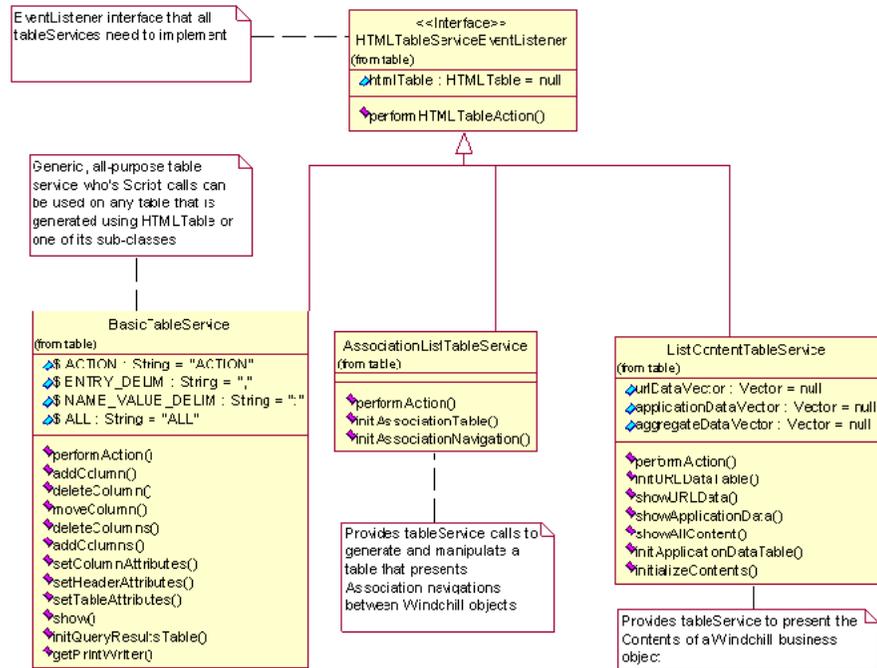
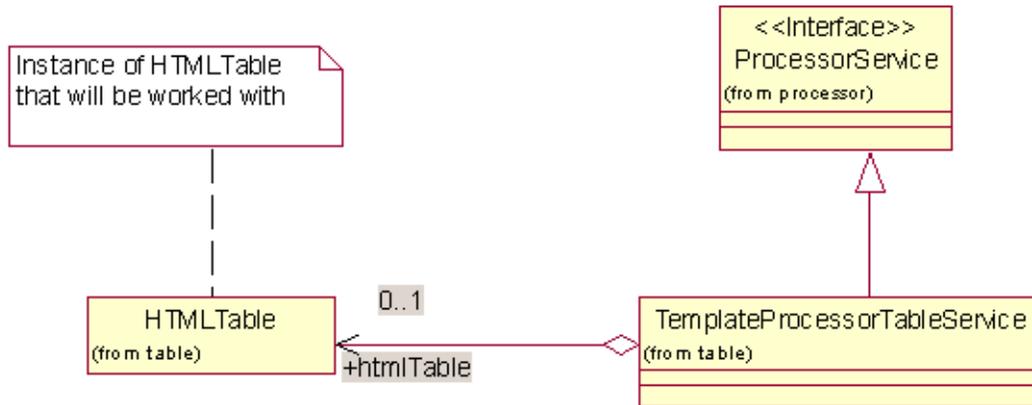
There are several different subclasses of `HTMLTable` with different functionality. There are also several different concrete implementations of the `Swing TableModel`. Each of these subclasses and combination of subclasses can have unique requirements on how it should be customized by the Windchill script. Consequently, the service must be extensible. In this case, extensible means being able to extend the service without having to change the existing source code.

The implementation used solved the two preceding design requirements by doing the following:

- Implementing the `wt.templateutil.processor.ProcessorService` interface. By implementing the `tableService` with this interface, the `tableService` is able to access the current state of the HTML Page being generated. This allows the service to act as if the service is implemented with the template processor while existing outside of the template processor. This prevents the addition of large amounts of code to `BasicTemplateProcessor` in order to allow all template processors access to this service.

- Using an event-based service that allows the addition of new listeners by encapsulating the Windchill script call into an event, a list of listeners can be notified of each script call and respond to it. The result is the ability to extend the tableService without having to effect the existing code. To add a new listener, you need only implement the HTMLTableServiceEventListener interface and register the new listener by an entry in wt.properties file for the entry, as follows:

```
wt.templateutil.table.HTMLTableServiceEventListeners=...
```





# 8

## Customizing PDMLink Template Processing Clients

As described in the chapter "Customizing PDMLink", most PDMLink HTML clients are based either on the template processing architecture or on jsp processing. This section describes some ways PDMLink pages based on HTML template processing may be customized. It assumes you are familiar with the template processing concepts described in the chapter Customizing the HTML Client and will describe some extensions that were made to the template processing technology to support PDMLink UI features and its look and feel..

<b>Topic</b>	<b>Page</b>
Directory Structure .....	8-2
Properties and Property Files .....	8-3
Template Processing Extensions for PDMLink .....	8-5
Creating a Table in a Template Processor .....	8-34
Common Customizations .....	8-46

## Directory Structure

The HTML templates, java files, and other files used in PDMLink template processing are primarily located in the following directories under WT\_HOME:

- codebase/templates/HTMLtemplateutil  
Templates and javascript common to multiple PDMLink pages -- for example, the files used to generate the page header, page footer, etc.
- codebase/com/ptc/core/HTMLtemplateutil/server/processor  
Processor and delegate classes used to generate and process HTML common to multiple PDMLink pages - for example, page header and footer, wizard components, and buttons
- codebase/templates/pdmlink
- Templates specific to each HTML client, organized by functional area
- codebase/com/ptc/windchill/pdmlink/\*/server/processors  
Processor and delegate classes used to generate and process PDMLink client pages, organized by functional area
- codebase/wt/templateutil/components  
Java classes for components and features that are used primarily for PDMLink --- for example, the PDMLinkHTMLTable and the actions dropdown menus, as well as classes used for Windchill PDM features.
- codebase/com/ptc/core/ui  
Contains the stylesheet solutions.css used for template processing pages
- codebase/com/ptc/core/ui/images  
codebase/wtcore/images  
codebase/wt/clients/images  
Image files used for PDMLink and other applications

If you must create new templates for your customizations, we suggest you put them in a new directory under <WT\_HOME>/codebase/templates such as <WT\_HOME>/codebase/templates/<MyCompany>, where MyCompany is a name of your choosing. This will prevent them from being deleted if you reinstall or upgrade PDMLink. Your site's template directory must be under the <WT\_HOME>/codebase/templates directory.

If you modify out-of-the-box templates you should keep a copy of your modified templates in a directory outside of <WT\_HOME>/codebase because files in codebase may be deleted if you reinstall or upgrade your PDMLink installation.

If you create new Java processors or delegates for your customizations, we suggest you put the source code in a new directory used only for your

customizations -- for example, `<WT_HOME>/src/com/<MyCompany>` where `<MyCompany>` is a name of your choosing.

The source files should be compiled to a directory in `<WT_HOME>/codebase` such as `<WT_HOME>/codebase/com/<MyCompany>`.

## Properties and Property Files

The properties configuring PDMLink out-of-the-box HTML clients are not as centrally located as those for Windchill PDM. In Windchill PDM, the file `codebase/service.properties` contains most of the entries mapping actions to `ActionDelegates`, `URLActionDelegates`, `FormTaskDelegates` and `TemplateProcessors`, and the file `HTMLtemplate.properties` contains most of the entries mapping actions to template files.

In PDMLink these property entries are distributed over multiple property files according to functional area. The directory `/codebase/com/ptc/windchill/pdmlink` contains most of the template processing code and much of the service code for PDMLink. That directory contains subdirectories for different functional area. Each functional area subdirectory typically contains a property file with template processing properties for that area.

For example, the file `codebase/com/ptc/windchill/pdmlink/doc/doc.properties` contains entries for HTML document clients, the file `codebase/com/ptc/windchill/pdmlink/change/change.properties` contains entries for HTML change management clients, etc. In addition the following files may be of interest:

- `codebase/com/ptc/windchill/pdmlink/templateutil/HTML.properties`  
Contains some properties for common components and main pages.
- `codebase/typedservices.properties`  
Contains properties for soft-type aware features, such as the Details page navigation bar and actions dropdown.

If you need to add or modify properties for your customizations, we recommend you put them in one or more new properties files as described in the section called `Properties and Property Files` in the chapter `Customizing the HTML Client`.

## Template Framework

In general, PDMLink makes greater use of subtemplates than Windchill PDM. The skeleton template for a typical PDMLink view page follows. Note that Windchill script method calls must be on one line in practice. They are shown on multiple lines below only for display in this document.

```
<html>
```

```

<head>

<!--The following includes the subtemplate
codebase/templates/HTMLtemplateutil/MetaInfo.html, which sets the
character encoding of the page based on the client browser's
language setting-->

<script language=Windchill>
<!--
    processSubTemplate action=GetMetaInfo
-->
</script>

<!--The following creates the <BASE> tag of the page. -->

<script language=Windchill>
<!--
    useProcessorService method=getBaseTag
service=com.ptc.core.HTMLtemplateutil.server.processors.UtilProces
sorService
-->
</script>

<!-- The following inserts style sheet links onto the page-->

<script language=windchill>
<!--
    useProcessorService method=getCSSLink
service=com.ptc.core.HTMLtemplateutil.server.processors.UtilProces
sorService
-->
</script>
<title><!-- your title goes here --></title>
</head>

<!--The following inserts some javascript on the page from
codebase/templates/HTMLtemplateutil/Javascript.html and creates a
<BODY> tag-->

<script language=windchill>
<!--
    useProcessorService method=getBodyTag
service=com.ptc.core.HTMLtemplateutil.server.processors.UtilProces
sorService
-->
</script>

<!--The following creates the page banner, including Learn, Help and
other links; search component, tabs, second-level navigation bar
under the current tab, and the context title bar under the second-
level navigation bar-->

<script language=windchill>
<!--
    processSubTemplate action=CreateBanner
-->
</script>

```

```
<!-- body of page goes here -->

<!--The following creates the page footer, including the navigation
links-->

<script language=windchill>
<!--
    processSubTemplate action=CreateFooter
-->
</script>

</body>
</html>
```

HTML wizards share some of this framework but do not have the banner and footer components.

## Template Processing Extensions for PDMLink

### Action Configuration

The file `codebase/wt/templateutil/NavigationAndActions.xml` defines the following lists of navigation links and actions:

- The links shown in the navigation bar on Details pages
- The actions available in the actions dropdown list on Details pages
- The actions available in the Actions column of a table for an object
- The actions available in the Multiselect actions header of a table
- The actions available in the Context-specific actions header of a table
- The tabs in wizards

Most of these features are shown in the following part Details page:



Windchill

Home | Product | Project | Change | Library | Organization | Site

Products List | Details | Folders | Product Structure | Team | Change Monitor | Forum | Workspaces | Templates | Utilities

GOLF\_CART Recent Products: [dropdown] [refresh] [add]

WHEEL\_ASSY [dropdown: ---Actions---] [Go]

**Responsible Product:** GOLF\_CART  
**Number:** GC000040  
**Name:** WHEEL\_ASSY  
**Version:** A.1  
**Type:** Separable  
**Source:** Make  
**View:** Manufacturing  
**Status:** Checked in  
**Location:** / GOLF\_CART / Manufacturing

**Team Name:** GC000040 - WHEEL\_ASSY A (Manufacturing) 1753  
**State:** In Work - Released - Cancelled  
**Created:** 2003-10-21 16:03:12 CDT  
**Created By:** wadmin  
**Updated By:** wadmin  
**Last Updated:** 2003-10-31 09:57:26 CST

**Product Structure** Configuration: [Latest 'Manufacturing' including work in progress] Related Reports: [Single Level BOM] [Generate Report]

Name	Actions	Number	Version	Context	State	Quantity	Line
<input type="checkbox"/> WHEEL_ASSY	[actions]	GC000040	A (Manufacturing)	GOLF_CART	In Work		
<input type="checkbox"/> AXLE_SLEEVE	[actions]	GC000034	A (Design)	GOLF_CART	In Work	1 each	
<input type="checkbox"/> BEARING_AXLE	[actions]	GC000033	A (Design)	GOLF_CART	In Work	1 each	
<input type="checkbox"/> HUB_CAP	[actions]	GC000037	A (Design)	GOLF_CART	In Work	1 each	
<input type="checkbox"/> TIRE	[actions]	GC000036	A (Design)	GOLF_CART	In Work	1 each	
<input type="checkbox"/> WHEEL_HUB	[actions]	GC000035	A (Design)	GOLF_CART	In Work	1 each	

Details Page Navigation Bar

Table Actions Column

Context Title Bar

Table Checkbox Column

Multiselect Table Actions

Context-Specific Table Actions

Table

Each list is contained in a NAVIGATION\_TREE section within NavigationAndActions.xml. For example the following tree gives the actions that may appear (depending on user permissions, object state, etc.) in the actions dropdown list on the Details page of a Problem Report.

```

<NAVIGATION_TREE>
  <Name>PR_Details_Actions</Name>
  <Links>
    <Link>
      <Name>Update_PR</Name>
      <Action>Update_PR</Action>
      <Tree_Action_Delegate/>
      <In_Minimum_List>>true</In_Minimum_List>
      <Resource_Bundle>
        com.ptc.windchill.pdmlink.change.server.processors.detailsResource
      </Resource_Bundle>
      <Resource_Key>0</Resource_Key>
      <Links/>
    </Link>
    <Link>
      <Name>Create_New_ECR</Name>
      <Action>Create_New_ECR</Action>
      <Tree_Action_Delegate/>
      <In_Minimum_List>>true</In_Minimum_List>
      <Resource_Bundle>
        com.ptc.windchill.pdmlink.change.server.processors.detailsResource
      </Resource_Bundle>
      <Resource_Key>2</Resource_Key>
      <Links/>
    </Link>
    <Link>
      <Name>Copy</Name>
      <Action>Copy</Action>
      <Tree_Action_Delegate/>
      <In_Minimum_List>>true</In_Minimum_List>
      <Resource_Bundle>
        com.ptc.core.HTMLtemplateutil.server.processors.processorsResource
      </Resource_Bundle>
      <Resource_Key>3</Resource_Key>
      <Links/>
    </Link>
    <Link>
      <Name>Delete_PR</Name>
      <Action>Delete_PR</Action>
      <Tree_Action_Delegate/>
      <In_Minimum_List>>true</In_Minimum_List>
      <Resource_Bundle>
        com.ptc.windchill.pdmlink.change.server.processors.detailsResource
      </Resource_Bundle>
      <Resource_Key>48</Resource_Key>
      <Links/>
    </Link>
  </Links>
</NAVIGATION_TREE>

```

```

    <Link>
      <Name>DISCUSS_PLM</Name>
      <Action>DISCUSS_PLM</Action>
      <Method>invokeAction</Method>
      <Tree_Action_Delegate/>
      <In_Minimum_List>>true</In_Minimum_List>
      <Resource_Bundle>
        com.ptc.windchill.pdmlink.change.server.processors.detailsResource
      </Resource_Bundle>
      <Resource_Key>55</Resource_Key>
      <Links/>
    </Link>
  </Links>
</NAVIGATION_TREE>

```

Each action or link in a navigation tree is enclosed in a <LINK> tag. The tags within a <LINK> node are as follows:

- Name - Used on HTML templates and in processor classes to reference a particular node. May be any descriptive name for the action, unique within the tree, but is typically the same as the action name.
- Action - The string used to look up an ActionDelegate and URLActionDelegate that will determine if the action should be listed for a specific object/user and create the URL for the link, respectively.
- Tree\_Action\_Delegate - currently not used
- Method (optional) - If your action invokes a template processing URL, this is the method in wt.enterprise.URLProcessor that should be called. The default is URLTemplateAction. See the section ActionDelegates and URLActionDelegates for more information.
- In\_Minimum\_List - currently not used
- Resource\_Bundle - The \*.rbInfo file that contains the string that should be displayed on the page for this link
- Resource\_Key - The number assigned to the string for this link in the \*.rbInfo file
- Links (second level) - currently not used

If you need to customize this file, you should copy the contents to a new file in your customization directory and then override the property `wt.templateutil.components.navigationbar.file` in `codebase/wt.properties` so that it points to your new file. To override this property you should include a line similar to the following within the `<Configuration>` tag in the file `<WT_HOME>/site.xconf`:

```
<Property name="wt.templateutil.components.navigationbar.file"
  overridable="true" targetFile="codebase/wt.properties"
  value="\$(wt.codebase.location)\$(dir.sep)com\$(dir.sep)
  MyCompany\$(dir.sep)NavigationAndActions.xml" />
```

where the attribute "value" is the path to your customized file. In this example, the path is `<WT_HOME>/codebase/com/MyCompany/NavigationAndActions.xml`.

Once you have added this tag, you must regenerate your site's property files using the command:

```
<WT_HOME>/bin/xconfmanager -p
```

This will set the property `wt.templateutil.components.navigationbar.file` in `codebase/wt.properties` to your customized `NavigationAndActions.xml` file.

## ActionDelegates and URLActionDelegates

Each of the actions or links referenced in the `NavigationAndActions.xml` file requires an `ActionDelegate` class and a `URLActionDelegate` class. The `ActionDelegate` class determines whether the given action should be available to a specific user in a specific context. The `URLActionDelegate` class is used to generate a URL for the action. These delegates were described in the chapter `Customizing the HTML Client`.

More specifically, the `ActionDelegates` and `URLActionDelegates` associated with actions in `NavigationsAndActions.xml` must implement the `wt.templateutil.processor.NavBarActionDelegate` interface and `wt.templateutil.processor.NavBarURLActionDelegate` interface, respectively.

The `NavBarActionDelegate` interface extends `wt.enterprise.ActionDelegate` and provides the following additional methods:

```
public void setState(HTTPState state)
```

Associates the `HTTPState` object from the template processor generating the link with the `ActionDelegate`.

```
public void setActionName(String action)
    throws WTPROPERTYVETOException
```

Sets the action attribute of the object.

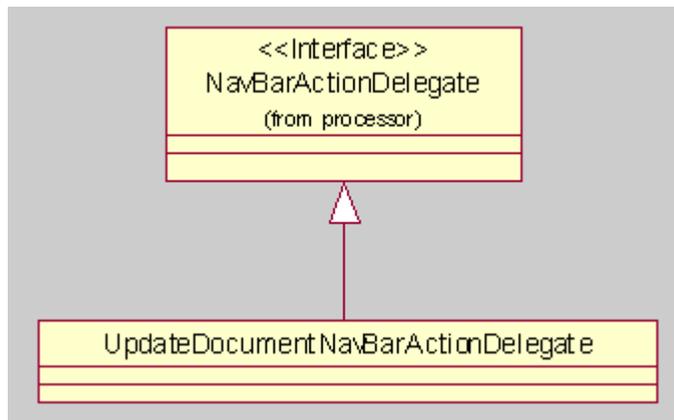
The NavBarActionDelegate is modeled. To create a NavBarActionDelegate for your action using Rational Rose 2002, perform the steps listed below. Alternatively, you could create your NavBarActionDelegate by hand based on the example class for Update Document shown below.

If you choose to create your delegate by hand, you would need to tailor your delegate to your situation by performing the customizations described in steps 10, 13, and 14 below.

To create a new NavBarActionDelegate using Rational Rose:

1. Open Rational Rose and load /src/WTDesigner.mdl or whatever .mdl file you are using for your customizations.
2. Go to the package in which you want to create your NavBarActionDelegate and create or open the class diagram in which you want to model your delegate.
3. Add a class to the class diagram and name it NavBarActionDelegate.
4. Add another class to the class diagram and give it the name of your new NavBarActionDelegate.
5. Add a Generalization arrow to specify that your new delegate implements NavBarActionDelegate.

Your Rose diagram should look similar to the following:



6. Open the Specification for your new delegate and go to the Windchill tab. Under Java Properties, set `Serializable` to `None`.
7. Generate the package.

8. Open the .java file generated for your new NavBarActionDelegate in an editor.
9. In the user.attributes section of the file add two variables as shown in bold below:

```


//##begin user.attributes preserve=yes
private HTTPState state = null;
private String action = null;
//##end user.attributes


```

10. In the static.initialization section of the file add one new variable as shown in bold below:

```


//##begin static.initialization preserve=yes
public static final String CONTEXT_SERVICE_NAME =
"<YOUR_ACTION_NAME>;"
//##end static.initialization


```

<YOUR\_ACTION\_NAME> should be the name of the action in the property used to look up the ActionDelegate and URLActionDelegate. In this file and in the property file, the name should all be in capital letters, regardless of its capitalization in the file NavigationAndActions.xml.

11. Complete the setState() method as shown in bold below:

```


public void setState( HTTPState state ) {
    //##begin setState%3C425C4B007Ds.body preserve=yes
    this.state = state;
    //##end setState%3C425C4B007Ds.body
}


```

12. Complete the setActionName() method as shown in bold below:

```


public void setActionName( String action )
    throws WTPPropertyVetoException {
    //##begin setActionName%3C425C600290s.body preserve=yes
    this.action = action;
    //##end setActionName%3C425C600290s.body
}


```

13. Modify the valid() method as appropriate for your action. This method should return true or false depending on whether the action is valid based on the type of the current object.
14. Modify the enableable() method as appropriate for your action. This method determines if the action should be enabled based on the current object's state and the current user's permissions.
15. Complete the getContextServiceName() method as shown in bold below:

```


public String getContextServiceName() {


```

```

        ///begin getContextServiceName%36DFF3B50157g.body
preserve=yes
        return CONTEXT_SERVICE_NAME;
        ///end getContextServiceName%36DFF3B50157g.body
    }

```

## NavBarActionDelegate for the Update Document Action Example

```

// Generated UpdateDocumentNavBarActionDelegate%3F89BF3400F2: Sun
10/12/03 15:58:45
/* bcwti
 *
 * Copyright (c) 2003 Parametric Technology Corporation (PTC). All
Rights
 * Reserved.
 *
 * This software is the confidential and proprietary information of
PTC.
 * You shall not disclose such confidential information and shall
use it
 * only in accordance with the terms of the license agreement.
 *
 * ecwti
 */

```

```
package com.ptc.windchill.pdmlink.doc.server.processors;
```

```

import java.lang.Boolean;
import java.lang.Object;
import java.lang.String;
import wt.templateutil.processor.HTTPState;
import wt.templateutil.processor.NavBarActionDelegate;
import wt.util.WTException;
import wt.util.WTPropertyVetoException;

```

```

///begin user.imports preserve=yes
import wt.access.AccessControlHelper;
import wt.access.AccessPermission;
import wt.doc.WTDocument;
import wt.session.SessionHelper;
import wt.vc.wip.Workable;
import wt.vc.wip.WorkInProgressHelper;
///end user.imports

```

```

///begin UpdateDocumentNavBarActionDelegate%3F89BF3400F2.doc
preserve=no

```

```

/**
 *

```

```

* <BR><BR><B>Supported API: </B>>false
* <BR><BR><B>Extendable: </B>>false
*
* @version 1.0
**/
/###end UpdateDocumentNavBarActionDelegate%3F89BF3400F2.doc

public class UpdateDocumentNavBarActionDelegate implements
NavBarActionDelegate {

    // --- Attribute Section ---

    private static final String RESOURCE =
"com.ptc.windchill.pdmlink.doc.server.processors.processorsResourc
e";
    private static final String CLASSNAME =
        UpdateDocumentNavBarActionDelegate.class.getName();

    /###begin user.attributes preserve=yes
        private HTTPState state = null;
        private String action = null;
    /###end user.attributes

    /###begin static.initialization preserve=yes
        public static final String CONTEXT_SERVICE_NAME =
            "UPDATEDOCUMENT";
    /###end static.initialization

    // --- Operation Section ---

    /###begin setState%3C425C4B007Ds.doc preserve=no
    /**
    * Sets the current HTTPState from the TemplateProcessor that is
being
    * used to generate the Navigation Bar.
    *
    * <BR><BR><B>Supported API: </B>>false
    *
    * @param state The current HTTPState.
    * @return void
    *
    **/
    /###end setState%3C425C4B007Ds.doc

    public void setState( HTTPState state ) {
        /###begin setState%3C425C4B007Ds.body preserve=yes
        this.state = state;
        /###end setState%3C425C4B007Ds.body
    }
}

```

```

//##begin setActionName%3C425C600290s.doc preserve=no
/**
 * Sets the current Action.
 *
 * <BR><BR><B>Supported API: </B>>false
 * <BR><BR><B>Supported API: </B>>false
 *
 *
 * <BR><BR><B>Supported API: </B>>false
 *
 * @param    action
 * @return   void
 *
 * @exception wt.util.WTPropertyVetoException
 **/
//##end setActionName%3C425C600290s.doc

public void setActionName( String action )
    throws WTPropertyVetoException {
    //##begin setActionName%3C425C600290s.body preserve=yes
    this.action = action;
    //##end setActionName%3C425C600290s.body
}

//##begin valid%3645CE5F005D.doc preserve=no
/**
 * Tests if the action is valid on all instances of the class of
object;
 * commonly used in the construction of dynamically generated
GUIs to
 * determine if a particular action should appear. For instance,
the
 * valid method of the "check-in" action will test if the object
is "Workable"
 * or not.
 *
 * <BR><BR><B>Supported API: </B>>false
 *
 * @param    object    The object that you want to test validity
on.
 * @return   Boolean
 **/
//##end valid%3645CE5F005D.doc

public Boolean valid( Object object ) {
    //##begin valid%3645CE5F005D.body preserve=yes

    return new Boolean(object instanceof WTDocument);
    //##end valid%3645CE5F005D.body
}

//##begin enableable%3645CEB702CE.doc preserve=no
/**
 * Tests if the action is currently applicable to the instance
object;

```

```

        * commonly used at the time a GUI is generated to see if the
action
        * should be "enabled" for the object, another common usage will
be at
        * the time that the action is to be performed, to see if it is
still
        * applicable. This method should not be called when the
concrete ActionDelegate
        * class is unknown. In that case, call
BasicTemplateProcessor.accessOK()
        * instead.
        *
        *
        * <BR><BR><B>Supported API: </B>>false
        *
        * @param    object    The object that you wish to set enabled.
        * @return    Boolean
        * @exception wt.util.WTException
        ** /
//##end enableable%3645CEB702CE.doc

public Boolean enableable( Object object )
    throws WTException {
    //##begin enableable%3645CEB702CE.body preserve=yes

    try {
        if (object instanceof Workable) {
            return new
Boolean(WorkInProgressHelper.isCheckedOut((Workable)object,
SessionHelper.manager.getPrincipal()));
        }
        else {
            return new
Boolean(AccessControlHelper.manager.hasAccess(object,
AccessPermission.MODIFY));
        }
    }
    catch (Exception e) {
        return new Boolean(false);
    }
}
//##end enableable%3645CEB702CE.body
}

//##begin getContextServiceName%36DFF3B50157g.doc preserve=no
/**
    * This method is to allow access to the variable,
CONTEXT_SERVICE_NAME.
    * The role of the variable CONTEXT_SERVICE_NAME is to provide a
reference
    * name of the Action being performed. The value of
CONTEXT_SERVICE_NAME
    * should be set in the subclass that implements this interface.
    *
    *
    * <BR><BR><B>Supported API: </B>>false
    *

```

```

    * @return    String
    **/
    ///end getContextServiceName%36DFF3B50157g.doc

    public String getContextServiceName() {
        ///begin getContextServiceName%36DFF3B50157g.body
        preserve=yes

        return CONTEXT_SERVICE_NAME;
        ///end getContextServiceName%36DFF3B50157g.body
    }

    ///begin setContextServiceName%36DFF7E101E4s.doc preserve=no
    /**
     * This method is to allow setting the variable,
     CONTEXT_SERVICE_NAME.
     * The role of the variable CONTEXT_SERVICE_NAME is to provide a
     reference
     * name of the Action being performed. The value of
     CONTEXT_SERVICE_NAME
     * should be set in the subclass that implements this interface.
     *
     * <BR><BR><B>Supported API: </B>>false
     *
     * @param    newContextServiceName
     **/
    ///end setContextServiceName%36DFF7E101E4s.doc

    protected void setContextServiceName( String
    newContextServiceName ) {
        ///begin setContextServiceName%36DFF7E101E4s.body
        preserve=yes

        ///end setContextServiceName%36DFF7E101E4s.body
    }

    ///begin user.operations preserve=yes
    ///end user.operations
}

```

The `NavBarURLActionDelegate` interface extends `wt.enterprise.URLActionDelegate` and provides the following additional methods:

```
public boolean isURLProvider(HashMap queryValueMap, Object object)
```

Given the name/value pairs from the query string of the URL, determines whether the URL was generated by the `URL()` method of this delegate.

```
public void setMethod(String method)
```

Sets the `wt.enterprise.URLProcessor` method to call in the URL.

```
public void setResourceBundleStr(String resourceBundleName)
```

Sets the name of the \*.rbInfo file to use for the label of the link.

```
public void setResourceKey(String key)
```

Sets the integer string of the link label in the \*.rbInfo file.

```
public void setState(HTTPState state)
```

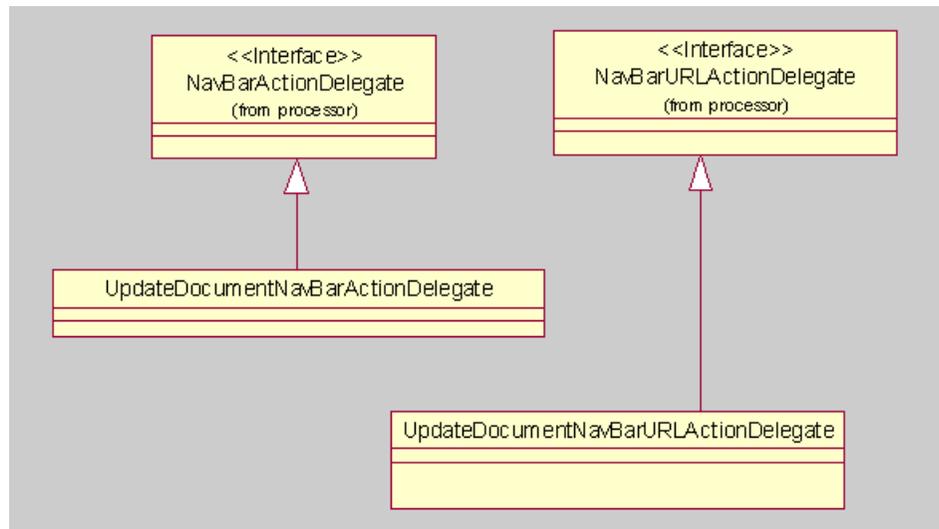
Associates the HTTPState object of the template processor generating the URL with the delegate.

The NavBarURLActionDelegate interface is modeled. To create a NavBarURLActionDelegate for your action using Rational Rose 2002, perform the following steps listed below. Alternatively, you could create your NavBarActionDelegate by hand based on the example class for the Update Document action shown below. You would need to make sure your delegate extends the appropriate NavBarActionDelegate and tailor your delegate to your situation by performing the customizations described in steps 13 and 19.

To create a new NavBarURLActionDelegate using Rational Rose:

1. Open Rational Rose and load /src/WTDesigner.mdl or whatever .mdl file you are using for your customizations.
2. Go to the package in which you want to create your NavBarURLActionDelegate and create or open the class diagram in which you want to model your delegate.
3. Add a class to the class diagram and name it NavBarURLActionDelegate.
4. Add another class to the class diagram and give it the name of your new NavBarURLActionDelegate.
5. Add a Generalization arrow to specify that your new delegate implements NavBarURLActionDelegate.
6. Add another class to the class diagram and give it the name of the NavBarActionDelegate for this action.
7. Add a Generalization arrow to specify that your new URLActionDelegate extends the NavBarActionDelegate for this action. If you were creating a

URLActionDelegate for the UpdateDocument action, after this step your Rose diagram should look similar to this this:



8. Open the Specification for your new delegate and go to the Windchill tab. Under Java Properties, set Serializable to None.
9. Generate the package.
10. Open the .java file generated for your new NavBarURLActionDelegate in an editor.
11. In the user.imports section of the file, import the class NavBarURLActionDelegateHelper shown in bold below:

```


###begin user.imports preserve=yes
import wt.templateutil.processor.NavBarURLActionDelegateHelper;
###end user.imports


```

12. In the user.attributes section of the file, add a new variable called helper shown in bold below:

```


###begin user.attributes preserve=yes
    protected NavBarURLActionDelegateHelper helper = null;
###end user.attributes


```

13. Replace the body of the method isURLProvider() with code appropriate to your action. The input argument query Value Map contains the name/value pairs from the query string of a URL (that is, the parameters following the ? in

the URL). Based on these query string values, your method should return true if they were generated by the URL() method of this delegate, or false if not. In some cases, the URL generated by a NavBarURLActionDelegate will be a template processing URL whose "action" parameter is the same as the Action tag value in the NAVIGATION\_TREE for this link. If that is the case, the body of the isURLProvider() method can be as shown in bold below:

```
public boolean isURLProvider( HashMap queryValueMap,
                               Object object ) {
    //##begin isURLProvider%3C425D9802AF.body preserve=yes
    return helper.isURLProvider(queryValueMap,object);
    //##end isURLProvider%3C425D9802AF.body
}
```

14. Complete the setResourceKey() method as shown in bold below:

```
public void setResourceKey( String key ) {
    //##begin setResourceKey%3C425DDE005Ds.body preserve=yes
    this.helper.setResourceKey(key);
    //##end setResourceKey%3C425DDE005Ds.body
}
```

15. Complete the setResourceBundleStr() method as shown in bold below:

```
public void setResourceBundleStr( String resourceBundle ) {
    //##begin setResourceBundleStr%3C425E15036Bs.body
    preserve=yes
    this.helper.setResourceBundleStr(resourceBundle);
    //##end setResourceBundleStr%3C425E15036Bs.body
}
```

16. Complete the setState() method as shown in bold below:

```
public void setState( HTTPState state ) {
    //##begin setState%3C425E5A02BFs.body preserve=yes
```

```
        this.helper.setState(state);  
        ///##end setState%3C425E5A02BFs.body  
    }
```

17. Complete the setMethod() method as shown in bold below:

```
public void setMethod( String method ) {  
    ///##begin setMethod%3C425E96002Es.body preserve=yes  
    this.helper.setMethod(method);  
    ///##end setMethod%3C425E96002Es.body  
}
```

18. Replace the `getActionName()` and `setActionName()` methods as shown in bold below:

```
public String getActionName() {
    //##begin getActionName%3649F3EC032Cg.body preserve=no
    return this.helper.getActionName();
    //##end getActionName%3649F3EC032Cg.body
}

public void setActionName( String a_ActionName )
    throws WTPPropertyVetoException {
    //##begin setActionName%3649F3EC032Cs.body preserve=no
    actionNameValidate( a_ActionName );
    this.helper.setActionName(a_ActionName);
    //##end setActionName%3649F3EC032Cs.body
}
```

19. Complete the `URL()` method so that it returns a URL string which will execute your action. In this method, you should call the `enableable()` method of the `NavBarActionDelegate` to check whether the current user is permitted to execute your action. If the action is not enabled, a URL value of null should be returned.

If your action should invoke a template processing page or delegate, you can use the `NavBarURLActionDelegate` helper to help you construct the URL as follows:

```
public String URL(Object object) throws WTEException {
    String gwURL = null;
    if (enableable(object).booleanValue()) {
        // Call the following method with true if you want an
        // "oid" parameter on the query string of the URL or
        // false, if not. The value of the oid will be a
        //reference to
        // the input object.
        helper.setOidAdded(true);
    }
}
```

```

        gwURL = helper.URL();
    }
    return gwURL;
}

```

The method above will produce a template processing url similar to the following:

```

http://<yourServerMachine>/<webappName>/servlet/WindchillAuthGW/
wt.enterprise.URLProcessor/URLTemplateAction?
action=<yourActionName>&oid=<reference to input object>

```

If you wanted to invoke the URLProcessor method invokeAction() instead of URLTemplateAction(), you could make the call helper.setMethod("invokeAction") before calling helper.URL() or include the following line in the <Link> section for this action in the NAVIGATION\_TREE:

```
<Method>invokeAction</Method>
```

If the URL to execute your action is not a simple template processing URL you will have to write your own method body to construct it. For example, if you want your action to call a javascript method to open a new window and launch a template processing Create Document wizard within that window, your URL() method might look something like this:

```

String jsUrl = null;
if (enableable(object).booleanValue()) {
    try {
        ReferenceFactory rf = new ReferenceFactory( );
        String oidStr =
            rf.getReferenceString((Persistable)object);
        URLFactory urlFactory =
            helper.getState().getURLFactory();
        HashMap map = new HashMap();
        map.put("action", "CreateDocument");
        map.put("oid", oidStr);
        // Url must be fully qualified for use in javascript
        // open window method so use
        //"buildAuthenticatedURL"
        // instead of "buildAuthenticatedHREF"
        URL url= GatewayServletHelper.buildAuthenticatedURL(
            urlFactory,
            "wt.enterprise.URLProcessor", "generateForm", map);
        String url = url.toExternalForm();
        JsUrl = "javascript:launchWindow(\" + url +
            "\", \"createDocWindow\", \
;width=600,height=400,\" +
            "directories=no,location=no,menubar=no,\" +
"scrollbars=yes,status=yes,toolbar=no,resizable=yes\"");
    }
}

```

```

        catch (Exception e) {
            throw new WTEException(CLASSNAME +
                ".URL() - exception creating link to " +
                "CreateDocument");
        }
    }
    return jsUrl;
}

```

The method above will produce a URL like the following:

```

javascript:launchWindow('http://<yourWebServerMachine>/
<yourWebAppName>/servlet/WindchillAuthGW/wt.enterprise.URLProcesso
r/generateForm?
action=CreateDocument&oid=OR%3Awt.pdmlink.PDMLinkProduct%3A187597&
u8=1',
'createDocWindow', 'width=600,height=400,directories=no,location=no
,menubar=no,
scrollbars=yes,status=yes,toolbar=no,resizable=yes')

```

**Note:** The above URL makes use of the javascript method "launchWindow()" which should already be included on most PDMLink template processing pages by virtue of calling the Windchill script method `getBodyTag()` and which can be used to launch a new window within a ProE embedded browser or a standalone browser.

20. Complete the `getURLLabel()` method as shown in bold below:

```

public String getURLLabel( Locale locale ) {
    return this.helper.getURLLabel(locale);
}

```

21. In the `user.operations` section of the file add a constructor method to the class as shown in bold below:

```

public CheckInDocumentNavBarURLActionDelegate() {
    helper = new NavBarURLActionDelegateHelper();
}

```



```

* @version 1.0
**/
/##end UpdateDocumentNavBarURLActionDelegate%3F89BFE5023A.doc

public class UpdateDocumentNavBarURLActionDelegate extends
UpdateDocumentNavBarActionDelegate implements
NavBarURLActionDelegate {

    // --- Attribute Section ---

    private static final String RESOURCE =

"com.ptc.windchill.pdmlink.doc.server.processors.processorsResource";
    private static final String CLASSNAME =
        UpdateDocumentNavBarURLActionDelegate.class.getName();
    private String actionName = null;
    private String classURL = null;

    /##begin user.attributes preserve=yes
protected NavBarURLActionDelegateHelper helper = null;
/##end user.attributes

/##begin static.initialization preserve=yes
/##end static.initialization

    // --- Operation Section ---

/##begin isURLProvider%3C425D9802AF.doc preserve=no
/**
 * Returns boolean flag indicating if the the current HashMap of
name
 * values pairs from the QueryString was generated by the current
 * NavBarURLActionDelegate.
 *
 * <BR><BR><B>Supported API: </B>>false
 *
 * @param queryValueMap The Name-Value pairs from the Query
String
 * @param object The target object used to generate the Url
 *
 * @return boolean
 *
 **/
/##end isURLProvider%3C425D9802AF.doc

    public boolean isURLProvider( HashMap queryValueMap, Object
object ) {
        /##begin isURLProvider%3C425D9802AF.body preserve=yes
        return helper.isURLProvider(queryValueMap,object);
        /##end isURLProvider%3C425D9802AF.body
    }
}

```

```

    ///begin setResourceKey%3C425DDE005Ds.doc preserve=no
    /**
     * Sets the integer reference of the entry to use in the Resource
    Bundle
     * specified be the getResourceBundleStr call
     *
     * <BR><BR><B>Supported API: </B>>false
     *
     * @param    key    The integer value of the key in the resource
    bundle
     *
     *
     * to use to generate localized label for the
    link
     *
     *
     */
    ///end setResourceKey%3C425DDE005Ds.doc

    public void setResourceKey( String key ) {
        ///begin setResourceKey%3C425DDE005Ds.body preserve=yes
        this.helper.setResourceKey(key);
        ///end setResourceKey%3C425DDE005Ds.body
    }

    ///begin setResourceBundleStr%3C425E15036Bs.doc preserve=no
    /**
     * Sets the name of the fully qualified classname of the resource
    bundle
     * to use. This will be the resource bundle used to generate the
    localized
     * text used for the getURLLabel method.
     *
     * <BR><BR><B>Supported API: </B>>false
     *
     * @param    resourceBundle    The target resource bundle to use
    generate
     *
     *
     * localized label for the link
     *
     *
     */
    ///end setResourceBundleStr%3C425E15036Bs.doc

    public void setResourceBundleStr( String resourceBundle ) {
        ///begin setResourceBundleStr%3C425E15036Bs.body
        preserve=yes
        this.helper.setResourceBundleStr(resourceBundle);
        ///end setResourceBundleStr%3C425E15036Bs.body
    }

    ///begin setState%3C425E5A02BFs.doc preserve=no
    /**
     * Sets the value of the current HTTPState object being used in
    the current
     * TemplateProcessor
     *
     * <BR><BR><B>Supported API: </B>>false
     *
     *
     */

```

```

    * @param    state    The current HTTPState instance
    *
    *
    **/
//##end setState%3C425E5A02BFs.doc

public void setState( HTTPState state ) {
    //##begin setState%3C425E5A02BFs.body preserve=yes
        this.helper.setState(state);
    //##end setState%3C425E5A02BFs.body
}

//##begin setMethod%3C425E96002Es.doc preserve=no
/**
 * Sets the method in URLAction to specify on the URL
 *
 * <BR><BR><B>Supported API: </B>>false
 *
 * @param    method    Name of a Gateway Method from
URLProcessor
 *
 *
 **/
//##end setMethod%3C425E96002Es.doc

public void setMethod( String method ) {
    //##begin setMethod%3C425E96002Es.body preserve=yes
        this.helper.setMethod(method);
    //##end setMethod%3C425E96002Es.body
}

//##begin getActionName%3649F3EC032Cg.doc preserve=no
/**
 * Gets the value of the attribute: actionName.
 *
 * <BR><BR><B>Supported API: </B>>false
 *
 * @return    String
 **/
//##end getActionName%3649F3EC032Cg.doc

public String getActionName() {
    //##begin getActionName%3649F3EC032Cg.body preserve=no

        return this.helper.getActionName();
    //##end getActionName%3649F3EC032Cg.body
}

//##begin setActionName%3649F3EC032Cs.doc preserve=no
/**
 * Sets the value of the attribute: actionName.
 *
 * <BR><BR><B>Supported API: </B>>false
 *
 * @param    a_ActionName
 * @exception wt.util.WTPPropertyVetoException
 **/

```

```

    ///end setActionName%3649F3EC032Cs.doc

    public void setActionName( String a_ActionName )
        throws WTPPropertyVetoException {
        ///begin setActionName%3649F3EC032Cs.body preserve=no

            actionNameValidate( a_ActionName );
            this.helper.setActionName(a_ActionName);
        ///end setActionName%3649F3EC032Cs.body
    }

    ///begin actionNameValidate%3649F3EC032C.doc preserve=no
    /**
     * @param      a_ActionName
     * @exception wt.util.WTPPropertyVetoException
     */
    ///end actionNameValidate%3649F3EC032C.doc

    private void actionNameValidate( String a_ActionName )
        throws WTPPropertyVetoException {
        if ( a_ActionName != null && a_ActionName.length() > 200 ) {
            Object[] args =
                {new wt.introspection.PropertyDisplayName(CLASSNAME,
"actionName"), "200"};
            throw new WTPPropertyVetoException(
                "wt.introspection.introspectionResource",
                wt.introspection.introspectionResource.UPPER_LIMIT,
args,
                new java.beans.PropertyChangeEvent(this, "actionName",
a_ActionName));
        }
    }

    ///begin getClassURL%3649F452000Fg.doc preserve=no
    /**
     * Gets the value of the attribute: classURL.
     *
     * <BR><BR><B>Supported API: </B>>false
     *
     * @return      String
     */
    ///end getClassURL%3649F452000Fg.doc

    public String getClassURL() {
        ///begin getClassURL%3649F452000Fg.body preserve=no

            return classURL;
        ///end getClassURL%3649F452000Fg.body
    }

    ///begin setClassURL%3649F452000Fs.doc preserve=no
    /**
     * Sets the value of the attribute: classURL.
     *
     * <BR><BR><B>Supported API: </B>>false

```

```

*
* @param      a_ClassURL
* @exception wt.util.WTPROPERTYVetoException
**/
//##end setClassURL%3649F452000Fs.doc

public void setClassURL( String a_ClassURL )
    throws WTPROPERTYVetoException {
    //##begin setClassURL%3649F452000Fs.body preserve=no

    classURLValidate( a_ClassURL ); // throws exception if not
valid
    classURL = a_ClassURL;
    //##end setClassURL%3649F452000Fs.body
}

//##begin classURLValidate%3649F452000F.doc preserve=no
/**
* @param      a_ClassURL
* @exception wt.util.WTPROPERTYVetoException
**/
//##end classURLValidate%3649F452000F.doc

private void classURLValidate( String a_ClassURL )
    throws WTPROPERTYVetoException {
    if ( a_ClassURL != null && a_ClassURL.length() > 200 ) {
        Object[] args = {new wt.introspection.PropertyDisplayName(
CLASSNAME,"classURL"),"200"};
        throw new WTPROPERTYVetoException(
            "wt.introspection.introspectionResource",
wt.introspection.introspectionResource.UPPER_LIMIT, args,
            new java.beans.PropertyChangeEvent(this,
"classURL",classURL,a_ClassURL));
    }
}

//##begin URL%3645CF0A0157.doc preserve=no
/**
* Returns a URL which when selected will perform the action on
the object.
* This method should not be called when the concrete
URLActionDelegate
* class is unknown. In that case, call
* BasicTemplateProcessor.getURLFromDelegate()
* instead.
*
* <BR><BR><B>Supported API: </B>>false
*
* @param      object
* @return     String
* @exception wt.util.WTException
**/
//##end URL%3645CF0A0157.doc

```

```

public String URL( Object object )
    throws WTEException {
    ///##begin URL%3645CF0A0157.body preserve=yes

    String jsUrl = null;
    if (enableable(object).booleanValue()) {
        try {
            if ( object instanceof Workable &&
(!WorkInProgressHelper.isWorkingCopy((Workable)object)) ) {
                object =
WorkInProgressHelper.service.workingCopyOf(
(Workable)object);
            }
            ReferenceFactory rf = new ReferenceFactory( );
            String oidStr =
rf.getReferenceString((Persistable)object);
            URLFactory urlFactory =
this.helper.getState().getURLFactory();
            HashMap map = new HashMap();
            map.put("action", "UpdateDocument");
            map.put("oid", oidStr);
            URL url=
GatewayServletHelper.buildAuthenticatedURL(urlFactory,
"wt.enterprise.URLProcessor", "generateForm", map);
            String gwURL = url.toExternalForm();
            String windowProperties =
"width=600,height=400,directories=no,location=no," +
"scrollbars=yes,status=yes,toolbar=no,resizable=yes";
            jsUrl = "javascript:launchWindow(\"' + url +
                ",\'UpdateDocWindow\',\'" + windowProperties +
                "\')";
        }
        catch (Exception e) {
            throw new WTEException(CLASSNAME +
                ".URL() - exception creating link to
UpdateDocument");
        }
    }
    return jsUrl;
    ///##end URL%3645CF0A0157.body
}

///##begin getURLLabel%3655D19701F4g.doc preserve=no
/**
 * Returns a localized label for a URL for the actions.
 *
 * <BR><BR><B>Supported API: </B>>false
 *
 * @param     locale
 * @return    String
 */
///##end getURLLabel%3655D19701F4g.doc

```

```

public String getURLLabel( Locale locale ) {
    ///##begin getURLLabel%3655D19701F4g.body preserve=yes
        return this.helper.getURLLabel(locale);
    ///##end getURLLabel%3655D19701F4g.body
}

///##begin user.operations preserve=yes
public UpdateDocumentNavBarURLActionDelegate() {
    helper = new NavBarURLActionDelegateHelper();
}
///##end user.operations
}

```

## Default ActionDelegates and URLActionDelegates

If your action should be available to all users in all contexts (that is no permission checking is needed), you do not need to write a NavBarActionDelegate for it. Instead, you can use the `wt.templateutil.processor.DefaultNavBarActionDelegate` class. You would add the following property to your system (see the section Properties and Property Files in the chapter Customizing the HTML Client for more information on how to add properties):

```

wt.services/svc/default/wt.enterprise.ActionDelegate/
<YOUR_ACTION>/java.lang.Object/0=
wt.templateutil.processor.DefaultNavBarActionDelegate/duplicate

```

If the url that should be generated for your action is a simple template processing url without an oid parameter on the query string, you can use the `wt.templateutil.processor.DefaultNavBarURLActionDelegate` for your action. You would add the following property for your action:

```

wt.services/svc/default/wt.enterprise.URLActionDelegate/
<YOUR_ACTION>/java.lang.Object/0=
wt.templateutil.processor.DefaultNavBarURLActionDelegate/
duplicate

```

If the url that should be generated for your action is a simple template processing url with an oid parameter referencing the current context object on the query string, you can use the `wt.templateutil.processor.ObjectPropsNavBarURLActionDelegate` class for your action. You would add the following property to your custom properties file for your action:

```

wt.services/svc/default/wt.enterprise.URLActionDelegate/
<YOUR_ACTION>/java.lang.Object/0=
wt.templateutil.processor.ObjectPropsNavBarURLActionDelegate/
duplicate

```

An example PDMLink table illustrating these components is shown below:

Multiselect Actions Bar
Context-specific Actions Bar

	Name	Actions	Number	Version	Context	State	Quantity	Line
<input type="checkbox"/>	LOWER_SUPPORT		GC000019	A (Design)	GOLF_CART	In Work		
<input type="checkbox"/>	LOWER_BAG HOLDER		GC000022	A (Design)	GOLF_CART	In Work	1 each	30
<input type="checkbox"/>	LOWER_SUPP_BAR		GC000020	A (Design)	GOLF_CART	In Work	1 each	10
<input type="checkbox"/>	PIN		GC000023	A (Design)	GOLF_CART	In Work	1 each	40
<input type="checkbox"/>	SUPPORT_CAP		GC000021	A (Design)	GOLF_CART	In Work	1 each	20

Remove | Add Alternate | Expand One | Expand All | Collapse | Show Documents | Hide Documents | Add to Baseline | Show Occurrences | Refresh

Multiselect Checkbox Column

Actions Column

Show/hide Column Icon

Problem Reports						
Name	Actions	Number	Change Admin	Priority	Requester	State
pr.2		00361	Administrators	Low	Administrator	Open
pr.1		00362	Administrators	Low	Administrator	Open

If using the DefaultNavBarURLActionDelegate or ObjectPropsNavBarURLActionDelegate and you would like the URL to invoke a URLProcessor method other than URLTemplateAction, you must specify the desired method in a <METHOD> tag for the <LINK> in NavigationAndActions.xml.

## Tables

The tables on PDMLink template processing pages are implemented using the same underlying components and services as those of Windchill PDM, described in the chapter Customizing the HTML Client in this manual. Some modifications and enhancements were made to those components for PDMLink, however. These include the following:

- Style changes to implement PDMLink look and feel

- A configurable actions column (optional)
- Sortable columns (optional)
- Show/hide columns (optional)
- A multiselect checkbox column and configurable multiselect actions bar (optional)
- A configurable context-specific actions bar (optional)

Click on the column name to sort the Actions column.

**Note:** In the out-of-the-box clients, tables are frequently enclosed within an expand-collapse section as follows:

In the picture above, the inner bracket on the right shows the extent of the table and the outer bracket shows the extent of the expand-collapse section. Creation of an expand-collapse section is not covered in this guide.

## Creating a Table in a Template Processor

Creation of a PDMLink table programmatically is very similar to creating a basic Windchill table. The major differences are as follows:

- Instead of instantiating a `wt.templateutil.table.HTMLTable` or `wt.templateutil.table.WtHtmlTable` as the controller class you instantiate a `wt.templateutil.table.PDMLinkHTMLTable`.
- The table model associated with the `PDMLinkHTMLTable` must implement `wt.templateutil.table.addColumn()` and `wt.templateutil.table.TableHeaderSetter()`. Any subclass of `wt.templateutil.table.RowDataTableModel` or `wt.templateutil.table.AssociationTableModel` will meet these requirements.
- The table model holding the data must be wrapped in a `wt.templateutil.table.SortedTableModel` to enable sorting on a column, if sorting is desired
- You must set the name attribute of the table to enable column sorting and collapsing, if sorting or collapsing is desired.
- Before calling the `init()` and `show()` methods of the table you must call the `SortedTableModel` to sort the rows, if sorting is desired
- You must include some javascript methods on your template for column sorting and collapsing (see [Sorting By Column](#) and [Column Collapsing](#) below).
- You must enclose your table within an HTML `<FORM>` element if your table has a checkbox column.

The following code is an example of how to create a PDMLinkHTMLTable in a template processor Windchill script method. It presents a table of the documents related to a part.

```

    public void displayRelatedDocsTable( Properties parameters,
    Locale locale, OutputStream os )
        throws WTEException {
        PrintWriter out = TemplateOutputStream.getPrintWriter(os,
    locale);
        WTPart part = (WTPart) getContextObj();

        // Acquire the objects to be displayed in the table rows
        QueryResult result =

    PartDocHelper.service.getAssociatedUpdateableDocuments(part);
        Vector rowDataObjects = new Vector();
        while (result.hasMoreElements()) {
            rowDataObjects.addElement(result.nextElement());
        }
        // Set the attributes (columns) to display for each object
        Vector columnVector = new Vector();
        columnVector.addElement("name");
        columnVector.addElement("number");
        columnVector.addElement("displayType");
        columnVector.addElement("containerName");
        columnVector.addElement("modifyTimestamp");

        // Create the table model
        RowDataTableModel rowDataTableModel = new
    RowDataTableModel();
        rowDataTableModel.setRowDataObjects(rowDataObjects);
        rowDataTableModel.setTableColumns(columnVector);
        rowDataTableModel.setLocale(locale);

        // Wrap it in a SortedTableModel so table can be
        // sorted by a given column
        SortedTableModel sortedModel = new SortedTableModel();
        sortedModel.setModel(rowDataTableModel);

        // Create the table
        PDMLinkHTMLTable table = new PDMLinkHTMLTable();
        table.setLocale(locale);
        table.setTableModel(sortedModel);
        table.setTableColumnModel( new DefaultHTMLTableColumnModel()
    );
        table.setPresentCheckBox(true); // set to false if no
    checkbox
                                           // column is desired

    table.setMultiSelectActionNavBarName("REMOVE_DOC_RELATION_ACTION")
    ;
        List multiSelectActionList = new ArrayList();
        multiSelectActionList.add("RemoveDocRelation");

```

```

        table.setMultiSelectActionList(multiSelectActionList);
        table.setFormName("RelatedDocumentsTable"); // name of the HTML
form element
        // containing this table. Used to
        // generate multiselect
                                                    // action links.
        table.setName("RelatedDocumentsTable"); // required for
column
                                                    // sorting and
collapsing
        table.setIsSortingEnabled(true); // set to false to
disable
                                                    // sorting by column
        table.setIsCollapsingEnabled(true); // set to false to
disable
                                                    // column collapsing
        table.setPresentIconFirstRow(true); // set to false if
don't want an icon
                                                    // for the object in
the first column
        table.setPresentActionsIconColumn(true); // set to false if
no actions
                                                    // column is desired
        table.createDefaultColumnsFromModel(); // must be called
after the above
                                                    // two methods

        table.setOutputStream(os);
        table.setState(getState());
        table.setFormData(parameters);

        // Sort the rows before displaying the table
        ColumnSortService sortService = new
ColumnSortService(getState());
        String columnName =
sortService.getSortedColumnName(table.getName());
        if (columnName != null)
        {
            int i =
table.getTableColumnModel().getColumnIndex(columnName);
            if (i > -1)
            {
                sortedModel.sortByColumn(i,
sortService.isSortAscending(table.getName(),
columnName));
            }
        }

        // Initialize the table
        table.init(null, null, null, null, null);

        // Display the table
        table.show(parameters);

        //getHTMLTableService().setHtmlTable( table );
        out.flush();
    }

```

The table produced by this code is shown below:

All		Remove				
	Name	Actions	Number	Type	Context Name	Last Updated
<input type="checkbox"/>	ASB Standards	 	0000000684	Document	GOLF_CART	2003-10-09 15:25:36 CDT
<input type="checkbox"/>	Manufacturing Plan	 	0000000682	Document	GOLF_CART	2003-10-09 15:24:09 CDT
<input type="checkbox"/>	Requirements Specification	 	0000000683	Document	GOLF_CART	2003-10-09 15:24:53 CDT

**Note:** This table has a multiselect actions bar with one action named Remove. It does not have a context-specific actions bar. If one were desired, it could be added in much the same way as the multiselect actions bar with code similar to the following:

```
table.setActionNavBarName("Name of tree in NavigationAndActions
file");
List actionList = new ArrayList();
actionList.add("Name of link in tree");
table.setActionList(actionList);
```

For either actions bar you select those links in a tree you want to display using the multiSelectActionList.add() or actionList.add() methods.

## Creating a Table from a Template

An easier way of creating a PDMLinkTable is to use a table service from the HTML template. With the table services, no Java coding is needed.

For example, assume you would like to display a table of all the alternates for a given part. Here is an example of how to construct such a table using the AssociationNavigationTableService.

```
<script LANGUAGE=Windchill>
<!--
tableService ACTION=initAssociationNavigation USEMASTER=true
ROLE=alternates LINKCLASSNAME=wt.part.WTPartAlternateLink
OTHERSIDECLASS=wt.part.WTPartMaster
tableService ACTION=initAssociationTable PDMLINKTABLE=true
NAME=AlternatesTable OTHERSIDEATTRIBUTES=number,name,containerName
USECHECKBOXTABLE=true
tableService ACTION=enableColumnCollapse isEnabled=true
tableService ACTION=enableSorting isEnabled=true
tableService ACTION=defineActionBar
MULTISELECTACTIONNAVBAR=Alternates_MultiSelectActions
MULTISELECTACTIONLIST=DeletePartAlternate
ACTIONNAVBAR=Alternates_ContextActions
ACTIONLIST=AddPartAlternate
tableService ACTION=setColumnAttributes NAME=number,name,containerName
```

```
td.ALIGN=left NOWRAP font.SIZE=2
tableService ACTION=setHeaderAttributes COLUMNNUMBER=all
th.ALIGN=left font.SIZE=3
tableService ACTION=setColumnAttributes COLUMNNUMBER=0 td.WIDTH=5%
tableService ACTION=setTableAttributes table.ALIGN=center table.WIDTH=100%
table.CELLSPACING=2 table.CELLPADDING=2 table.BORDER=0
tableService ACTION=setHeaderFromResource COLUMNNAME=containerName

RESOURCEBUNDLE=com.ptc.windchill.pdmlink.part.server.processors.processorsResource
RESOURCEKEY=CONTEXT
tableService ACTION=show
-->
</script
```

The resulting table is shown below:

All		Remove		Add Alternate	
	Number	Actions	Name	Context	
<input type="checkbox"/>	0000000971	①	Alternate Lower Support A	GOLF_CART	
<input type="checkbox"/>	0000000972	①	Alternate Lower Support B	GOLF_CART	

The things that distinguish the above sequence of table service calls from a basic Windchill table are:

- The attribute PDMLINKTABLE for the initAssociationTable action indicates a PDMLinkHTMLTableModel should be used
- The USECHECKBOXTABLE, as in Windchill, indicates a checkbox column should be added to the table. For PDMLink, this column will have an "All" checkbox at the top.
- The action "enableColumnCollapse" enables collapsing of columns. Column collapsing is on by default so this action call can be omitted if you want collapsing enabled. If you want to disable column collapsing, you must include this call and set isEnabled to false.
- The action "enableSorting" enables sorting by column. Sorting is on by default so this action call can be omitted if you want sorting enabled. If you want to disable sorting, you must include this call and set isEnabled to false.
- The action "defineActionBar" defines the multiselect actions and context-specific actions to be displayed on the actions bar at the top of the table
- The "name" attribute for the call to initAssociationTable provides a unique name for the table that is used for column sorting and hiding
- As when creating a table programmatically, if you enable column sorting and collapsing you must include some javascript on your template (see Sorting By Column and Column Collapsing below).
- You must enclose your table within an HTML <FORM> element if your table has a checkbox column.

The attributes defined for the defineActionBar action are:

- multiSelectActionNavBar - the name of the <NAVIGATION\_TREE> in NavigationAndActions.xml that defines the multiselect action links for this table
- multiSelectActionList - the <NAME> tag values for the <LINK> actions in the <NAVIGATION\_TREE> you want to display in this table. Names should be comma-separated.

- `actionNavBar` - the name of the `<NAVIGATION_TREE>` in `NavigationAndActions.xml` that defines the context-specific action links for this table
- `actionList` - comma separated list of `<NAME>` tag values for the `<LINK>` actions in the `<NAVIGATION_TREE>` you want to display in this table. Names should be comma-separated.
- `formName` - name of the HTML form element containing the table. This is used to construct URLs for the multiselect actions in the table.

If you omit the table service call to `defineActionBar`, the action bar will be omitted from the table.

Here is an example of using the `ListContentTableService` to display a table of the `ApplicationData` contents of a `ContentHolder` object:

```
<SCRIPT LANGUAGE=Windchill>
<!--
    tableService ACTION=initializeContents
    tableService ACTION =initapplicationdatatable
        APPLICATIONDATAATTRIBUTES=
            fileName,format,fileSize
        name=AttachmentsTable PDMLINKTABLE=true
        SUPPRESSACTION=true
tableService ACTION =setHeaderFromResource POSITION=0
    RESOURCEBUNDLE=
        com.ptc.windchill.pdmlink.doc.server.
        processors.processorsResource
    RESOURCEKEY=FILE_NAME
    tableService ACTION =setHeaderFromResource POSITION=1
    RESOURCEBUNDLE=
        com.ptc.windchill.pdmlink.doc.client.
        clientRB
    RESOURCEKEY=FORMAT
tableService ACTION =setHeaderFromResource POSITION=2
    RESOURCEBUNDLE=
        com.ptc.windchill.pdmlink.doc.server.
        processors.processorsResource
    RESOURCEKEY=FILE_SIZE
tableService ACTION =setHeaderFromResource POSITION=2
    RESOURCEBUNDLE=
        com.ptc.windchill.pdmlink.doc.server.
        processors.processorsResource
    RESOURCEKEY=UPDATE_STAMP
tableService ACTION =setHeaderFromResource POSITION=2
    RESOURCEBUNDLE=
        com.ptc.windchill.pdmlink.doc.server.
        processors.processorsResource
    RESOURCEKEY=CREATOR
tableService ACTION =setTableAttributes table.WIDTH=100%
tableService ACTION =show
```

```
-->
</SCRIPT>
```

The table produced looks like the following:

File name	Format	File size	Last modified	Created by
 AphelionAdminGuide.pdf	PDF	2418.3 KB	2003-11-06 10:15:37 CST	wcadmin
 ESIUsersGuide.pdf	PDF	1579.0 KB	2003-11-06 10:15:40 CST	wcadmin
 PDMAguide.pdf	PDF	1108.95 KB	2003-11-06 10:15:43 CST	wcadmin

**Note:** In this table, the Actions column was deleted by the SUPPRESSACTION=true attribute on the initApplicationDataTable call.

Also the no actions bar is presented at the top of the table because there is no call to defineActionBar.

## Multiselect Action Bar

The actions presented on the multiselect action bar require knowledge of which checkboxes are checked. The typical model for such actions is as follows:

- The table is included within an HTML form element, the name of which is communicated to the PDMLinkHTMLTable by the formName attribute on the defineActionBar table service call or the java method PDMLinkHTMLTable.setFormName()
- The NavBarURLActionDelegate that should be used for actions on the multiselect action bar is wt.templateutil.processor.ActionNavBarURLActionDelegate. The URL() method of this delegate will generate a Javascript URL of the following form:

```
submitMultiSelectAction(String formName,String URL)
```

The form name argument is taken from the PDMLinkTable formName attribute defined above. The URL is a simple template processing URL like the following:

```
.../wt.enterprise.URLProcessor/  
URLTemplateAction?action=<action name>&  
oid=<reference to current context object>
```

where:

action is the name given to your action in the NAVIGATION\_TREE defining the action bar and that used to look up your NavBarActionDelegate and NavBarURLActionDelegate

oid is an object reference to the context object for the page containing the table

The gateway method used in the URL will be URLTemplateAction unless you specify an alternative method in the <LINK> tag for this action in the NAVIGATION\_TREE for the action bar. For example, for the <LINK> tag:

```
<Link>
    <Name>RemoveDocumentRelation</Name>
    <Action>RemoveDocRelation</Action>
    <Method>processForm</Method>
    <Tree_Action_Delegate/>
    <In_Minimum_List>true</In_Minimum_List>
    <Resource_Bundle>
        com.ptc.windchill.pdmlink.change.client.changeRB
    </Resource_Bundle>
    <Resource_Key>21</Resource_Key>
    <Links/>
</Link>
```

The following URL would be generated:

```
.../wt.enterprise.URLProcessor/processForm?action=RemoveDocRelation&
oid=<reference to current context obj>
```

If the URLProcessor method used is URLTemplateAction or generateForm, the system will expect to find a property mapping the action to a template and template processor. If the URLProcessor method used is processForm or invokeAction, the system will expect to find a property mapping the action to a FormTaskDelegate.

- submitMultiSelectAction refers to a javascript method in the file <WT\_HOME>/codebase/templates/HTMLtemplateutil/submitFunctions.js. This javascript method is available to most PDMLink pages by virtue of a call to UtilProcessorService.getBodyTag(). The method is as follows:

```
function submitMultiSelectAction(formName,formAction)
{
    var submitForm;
    for (var i = 0; i < document.forms.length; i++)
    {
        if (document.forms[i].name == formName)
        {
            submitForm = document.forms[i];
        }
    }
    submitForm.action = formAction;
    submitForm.submit();
}
```

- When a multiselect action is invoked, the URL passed to submitMultiSelectAction will be invoked and the form data, including checkboxes, will be passed to the method on the URL.

The template processor or FormTaskDelegate that handles the action will have the form data available to it in its associated HTTPState object. It can be assessed through the processor's or delegate's getFormData() method. The selected checkboxes can be identified by form data elements named as follows:

```
checkboxbox_<reference to table row object>
```

For example: checkbox\_VR:wt.part.WTPart:378972

## Actions Column

By default, an Actions column will be added to PDMLinkHTMLTables. If the first column of the table is an object icon column, the actions column will be the second column in the table. Otherwise, it will be the first column.

An Actions column only makes sense if each row of the table represents a PDMLink object for which you would like to make actions available from the table. If you do not want an Actions column to appear, you can do so programmatically by calling the following method:

```
PDMLinkHTMLTable.setPresentActionsIconColumn(false)
```

If you are creating a table using the AssociationListTableService or ListContentTableService you can remove the actions column by adding the attribute SUPPRESSACTION=true to the table initialization action. For example:

```
tableService action=initAssociationTable
otherSideAttributes=
```

```
name,number,team:CHANGE_ADMINISTRATOR_I,  
issuePriority,requester,lifecycleState  
sort=1 asc=false name=RelatedPRsTable PDMLINKTABLE=true  
SUPPRESSACTION=true
```

```
tableService action=INITAPPLICATIONDATATABLE  
APPLICATIONDATAATTRIBUTES=fileName,format,fileSize  
sort=0 asc=false name=AttachmentsTable PDMLINKTABLE=true  
SUPPRESSACTION=true
```

The actions that are displayed in the actions column depend on the type of object presented in the table row. A property file entry with the service name `wt.templateutil.icon.TableActions` maps a given object type to a tree in the `NavigationAndActions.xml` file containing the actions to be presented for that type. For example, the following entry specifies that the tree named `WTDOCUMENT_TABLE_ACTIONS` in `NavigationAndActions.xml` contains the actions to be listed in the actions column for `WTDocuments`:

```
wt.services/rsc/default/wt.templateutil.iconTableActions/  
TABLEACTIONS/wt.doc.WTDocument/0=WTDOCUMENT_TABLE_ACTIONS
```

Like other service lookups based on object type, if a mapping is not found for a given type the system will look recursively for a mapping for the parent type. If a mapping for a lower level parent is not found, an object inheriting from `WTOject` will receive the actions list specified by the tree `WTOBJECT_TABLE_ACTIONS` as a result of the of the following entry in the file `codebase/com/ptc/windchill/pdmlink/templateutil/HTML.properties`:

```
wt.services/rsc/default/wt.templateutil.iconTableActions/  
TABLEACTIONS/wt.fc.WTOject/0=WTOBJECT_TABLE_ACTIONS
```

The section [How to Add an Action to the Actions Column of a PDMLink Table](#) explains in more detail how to modify the actions displayed for a given object type.

## Sorting by Column

If you have enabled sorting by column in your table, you must include the following javascript and form element on the template that displays the table:

```
<script language=JavaScript>  
<!--  
function submitColumnSort( action )  
{  
    var cform = window.document.forms.sortby;  
    cform.sortby.value=action;  
    cform.submit();  
}  
-->  
</script>  
  
<Form action="<script language=Windchill><!-- getCurrentUrl --  
></script>"  
    method="post" name="sortby">  
    <input type="hidden" name="sortby" value="dummy">  
</Form>
```

The form element must be placed outside of any other form element on the page.

## Column Collapsing

If you have enabled column collapsing in your table, you must include the following javascript and form element on the template that displays the table:

```
<script language=JavaScript>
<!--
function submitColumnCollapse( action )
{
    var cform = window.document.forms.collapse;
    cform.collapse.value=action;
    cform.submit();
}
-->
</script>

<Form action="<script language=Windchill><!-- getCurrentUrl --
></script>"
    method="post" name="collapse">
    <input type="hidden" name="collapse" value="dummy">
</Form>
```

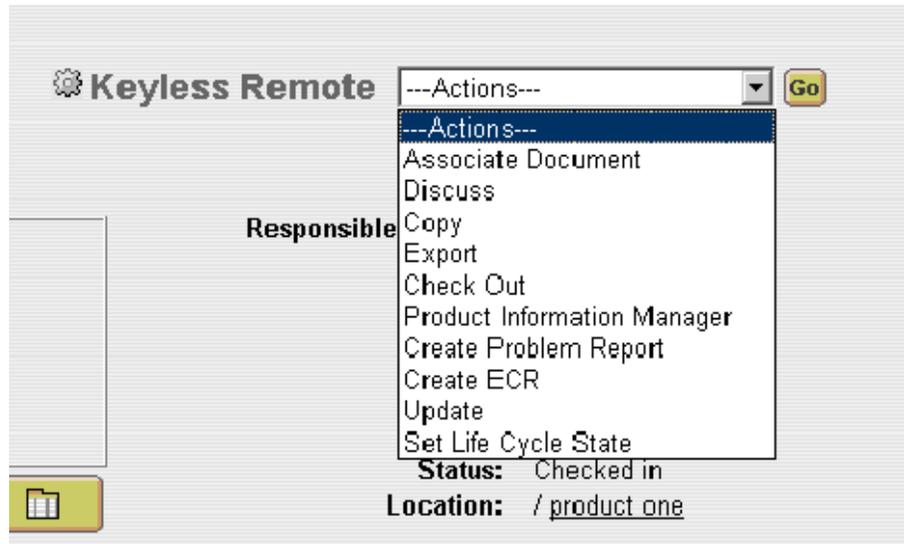
The form element must be placed outside of any other form element on the page.

## Common Customizations

### How to Add an Action to the Actions Dropdown Menu on a PDMLink Details Page

This section gives an example of how to add an action to the actions dropdown menu on a Details page. Assume that you would like to add the action Revise to the dropdown menu for parts after the Associate Document action (see below).

The Revise action should create a new revision of the part. No user input is needed for the revision.



Perform the following steps to add this action:

1. Create an entry in an .rbInfo file for the label for this action. For example, assume you have a .rbInfo file called /src/com/MyCompany/CustomResource.rbInfo containing 26 of your custom UI text strings. You would add an entry to that file similar to the following:

```
<27>.value=Revise  
  
<27>.constant=REVISE  
  
<27>.comment=Label for the Revise action in the Part actions  
dropdown list
```

2. Build your .rbInfo file by typing:

```
ResourceBuild com.MyCompany.CustomResource
```

This will generate CustomResource\*.class files in the  
<WT\_HOME>/codebase/com/MyCompany directory.

3. If you have not already done so, create a customized version of the file <WT\_HOME>/codebase/wt/templateutil/NavigationAndActions.xml as described in the section Actions Configuration above.

4. Discover the name of the NAVIGATION\_TREE containing the actions dropdown list for the WTPart Details page. Because the actions dropdown lists are soft-type-specific, you locate this information in the file /codebase/typedservices.properties. Search for the string "DROPDOWN" and then locate the entry containing "DROPDOWN" for wt.part.WTPart:

```
wt.services/rsc/default/wt.templateutil.objectPropsActionDropDown/DROPDOWN/wt.part.WTPart/0=WTPART_DROPDOWN_ACTIONS
```

At the end of this property entry is the name of the tree containing the Actions dropdown list for WTParts: "WTPART\_DROPDOWN\_ACTIONS."

5. Add the Revise action to the WTPART\_DROPDOWN\_ACTIONS tree in your custom NavigationAndActions.xml file as follows. Note that your action name must be unique within your PDMLink system.

```
<NAVIGATION_TREE>
<Name>WTPART_DROPDOWN_ACTIONS</Name>
<Links>
  <Link>
    <Name>Tools</Name>
    <Action>Tools</Action>
    <Tree_Action_Delegate/>
    <In_Minimum_List>true</In_Minimum_List>
    <Resource_Bundle>
      com.ptc.windchill.pdmlink.doc.server.serverResource
    </Resource_Bundle>
    <Resource_Key>25</Resource_Key>
    <Links/>
  </Link>
  <Link>
    <Name>ASSOCIATEDDOCS</Name>
    <Action>ASSOCIATEDDOCS</Action>
    <Tree_Action_Delegate/>
    <In_Minimum_List>true</In_Minimum_List>
    <Resource_Bundle>
      com.ptc.windchill.pdmlink.doc.server.serverResource
    </Resource_Bundle>
    <Resource_Key>18</Resource_Key>
    <Links/>
  </Link>
  <Link>
    <Name>Revise</Name>
    <Action>CustomRevise</Action>
    <Method>invokeAction</Method>
    <Tree_Action_Delegate/>
    <In_Minimum_List>true</In_Minimum_List>
    <Resource_Bundle>
      com.MyCompany.CustomResource
    </Resource_Bundle>
    <Resource_Key>27</Resource_Key>
    <Links/>
  </Link>
  .
</Links>
```

6. Create `src/custom/ReviseNavBarActionDelegate.java` and `src/custom/ReviseNavBarURLActionDelegate.java` classes for your action following the instructions in the section of this document called `ActionDelegates` and `URLActionDelegates`.

7. Create new property file entries so the system can find the `ActionDelegate` and `URLActionDelegate` associated with your action. Note that the action name must be upper case in your property entries. See the section titled `Properties and Property Files` in the chapter `Customizing HTML Client` for how to add new properties.

```
wt.services/svc/default/wt.enterprise.ActionDelegate/CUSTOMREVI  
SE/java.lang.Object/0=custom.ReviseNavBarActionDelegate/duplica  
te
```

```
wt.services/svc/default/wt.enterprise.URLActionDelegate/CUSTOMR  
EVISE/
```

```
java.lang.Object/0=custom.ReviseNavBarURLActionDelegate/duplica  
te
```

8. Create a `FormTaskDelegate` with a `processAction()` method to execute your `Revise` action as well as the necessary property file entry to map your action name to your `FormTaskDelegate`. `FormTaskDelegates` are described in the previous chapter "`Customizing the HTML Client`."
9. Restart your Method Server. You should now see a `Revise` item in the dropdown menu.

## How to Create a Soft-Type-Specific Actions Dropdown Menu

The `delegate/tree` lookup service for the actions dropdown menus is type-based, meaning different action lists can be specified for different soft types of a modeled object type. You do this by giving the soft type a different `NAVIGATION_TREE` list for the dropdown in your `NavigationAndActions.xml` file. If you do not specify a `NAVIGATION_TREE` specific to a soft type, that soft type will inherit the `NAVIGATION_TREE` of its modeled parent type, if one exists. If no tree for the parent type exists, the service will look for an entry for the parent type of the parent, and so on up the inheritance tree.

As an example of how to create an actions dropdown menu for a specific soft type, assume you have a soft type of `WTDocument` called `Specification Document`. Because this is a subtype of `WTDocument`, the `Details` page, including actions dropdown, for each instance will be identical to that for `WTDocument` without any action on your part. If you would like to modify the actions dropdown so that the `Set Life Cycle Action` is not shown for `Specification Documents` you would do the following:

1. Discover the name of the NAVIGATION\_TREE containing the actions dropdown list for the WTDocument Details page. You can locate this information in the file <WT\_HOME>/codebase/typedservices.properties. Search for the string "DROPDOWN" and then locate the entry containing "DROPDOWN" for wt.doc.WTDocument:

```
wt.services/rsc/default/wt.templateutil.objectPropsActionDropDown/
DROPDOWN/
wt.doc.WTDocument/0=WTDOCUMENT_DROPDOWN_ACTIONS
```

At the end of this property entry is the name of the tree containing the Actions dropdown list for WTDocuments:  
"WTDOCUMENT\_DROPDOWN\_ACTIONS."

2. If you have not already done so, create a customized version of the file <WT\_HOME>/codebase/wt/templateutil/NavigationAndActions.xml as described in the section Actions Configuration above.
3. Open your custom NavigationAndActions.xml file and copy the NAVIGATION\_TREE named WTDOCUMENT\_DROPDOWN\_ACTIONS. Paste the copy into your file and rename the copy SPECIFICATION\_DOCUMENT\_DROPDOWN\_ACTIONS. Your new tree may be placed anywhere in the file as long as it is between existing trees.
4. Go to the bottom of the tree for SPECIFICATION\_DOCUMENT\_DROPDOWN\_ACTIONS and delete the <Link> section named SetLifecycleState. Your file should now look similar to the following:

```
<NAVIGATION_TREE>
<Name>WTDOCUMENT_DROPDOWN_ACTIONS</Name>
  <Links>
    <Link>
      <Name>DISCUSS_PLM</Name>
      <Action>DISCUSS_PLM</Action>
      <Method>invokeAction</Method>
      <Tree_Action_Delegate/>
      <In_Minimum_List>true</In_Minimum_List>
      <Resource_Bundle>
        com.ptc.windchill.pdmlink.doc.server.serverResource
      </Resource_Bundle>
      <Resource_Key>19</Resource_Key>
    <Links/>
  </Link>

  <Link>
```

```

        <Name>RenameObject</Name>
        <Action>RenameObject</Action>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>wt.doc.docResource</Resource_Bundle>
        <Resource_Key>90</Resource_Key>
        <Links/>
    </Link>
    .
    .
    .
    <Link>
        <Name>EXPORT</Name>
        <Action>EXPORT</Action>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>
            com.ptc.windchill.pdmlink.clients.export.
            server.processors.processorsResource
        </Resource_Bundle>
        <Resource_Key>1</Resource_Key>
        <Links/>
    </Link>
    <Link>
        <Name>SetLifeCycleState</Name>
        <Action>SETLIFECYCLESTATE</Action>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>
            com.ptc.windchill.pdmlink.lifecycle.server.
            processors.lifecycleResource
        </Resource_Bundle>
        <Resource_Key>184</Resource_Key>
        <Links/>
    </Link>
    </Links>
</NAVIGATION_TREE>
<NAVIGATION_TREE>
<Name>SPECIFICATION_DOCUMENT_DROPDOWN_ACTIONS</Name>
<Links>
    <Link>
        <Name>DISCUSS_PLM</Name>
        <Action>DISCUSS_PLM</Action>
        <Method>invokeAction</Method>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>
            com.ptc.windchill.pdmlink.doc.server.serverResource

```

```

        </Resource_Bundle>
        <Resource_Key>19</Resource_Key>
        <Links/>
    </Link>

    <Link>
        <Name>RenameObject</Name>
        <Action>RenameObject</Action>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>wt.doc.docResource</Resource_Bundle>
        <Resource_Key>90</Resource_Key>
        <Links/>
    </Link>
    .
    .
    .
    <Link>
        <Name>EXPORT</Name>
        <Action>EXPORT</Action>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>
            com.ptc.windchill.pdmlink.clients.export.
            server.processors.processorsResource
        </Resource_Bundle>
        <Resource_Key>1</Resource_Key>
        <Links/>
    </Link>
</Links>
</NAVIGATION_TREE>

```

5. Create a typed services property entry mapping your soft type name to your new actions dropdown tree:

```

wt.services/rsc/default/wt.templateutil.objectPropsActionDropDown/
DROPDOWN/
WCTYPE/wt.doc.WTDocument/SpecificationDocument/0=SPECIFICATION_DOC
UMENT_DROPDOWN_ACTIONS

```

See the section called Properties and Property Files in the chapter Customizing the HTML Client for more information about adding new property entries.

6. Restart your Method Server. Details pages for Specification Documents should now display an actions dropdown list without a Set Life Cycle State action.

## How to Add a Link to a Details Page Navigation Bar

This section gives an example of how to add a link to the navigation bar on a Details page. Assume you would like to replace the Versions and Iteration History links with the single link "All Versions and Iterations" on the navigation bar on the Details page for documents (see below). When clicked, the new link will display a page with two tables: the Versions table from the Versions Details page and the Iteration History table from the Iteration History Details page.



To add this link, follow the steps below:

1. Create an entry in an .rbInfo file for the label for this link. For example, assume you have a .rbInfo file called `<WT_HOME>/src/com/MyCompany/CustomResource.rbInfo` containing 27

of of your custom UI text strings. You would add an entry to that file similar to the following:

```
28.value=All Versions and Iterations  
28.constant=VERSIONS_AND_ITERATIONS  
28.comment=Label for the All Versions and Iterations link on the  
WTDocument Details page
```

2. Build your .rbInfo file by typing:

```
ResourceBuild com.MyCompany.CustomResource
```

This will create CustomResource\*.class files in the directory  
<WT\_HOME>/codebase/com/MyCompany.

3. If you have not already done so, create a customized version of the file  
<WT\_HOME>/codebase/wt/templateutil/NavigationAndActions.xml as  
described in the section Actions Configuration above.

4. Discover the name of the NAVIGATION\_TREE containing the navigation  
bar for WTDocuments. You can locate this information in the file  
<WT\_HOME>/codebase/typeservices.properties. Search for the string  
"NAVBAR" and then locate the entry containing "NAVBAR" for  
wt.doc.WTDocument:

```
wt.services/rsc/default/wt.templateutil.objectPropsNavBar/NAVBAR/  
wt.doc.WTDocument/0=WTDOCUMENT_NAVBAR_ACTIONS
```

The name of the tree containing the navigation bar list is at the end of this  
entry: WTDOCUMENT\_NAVBAR\_ACTIONS.

5. Delete the Versions and Iteration History links and add a Versions and  
Iteration History link to the WTDOCUMENT\_NAVBAR\_ACTIONS tree in  
your custom NavigationsAndActions.xml file as follows. Note that your new  
action name must be unique within your PDMLink system.

```
<NAVIGATION_TREE>  
  <Name>WTDOCUMENT_NAVBAR_ACTIONS</Name>  
  <Links>  
    .  
    .  
    .  
  <Link>  
    <Name>ViewBaselinesForObject_plm</Name>  
    <Action>ViewBaselinesForObject_plm</Action>  
    <Tree_Action_Delegate/>  
    <In_Minimum_List>true</In_Minimum_List>  
    <Resource_Bundle>  
      com.ptc.windchill.pdmlink.baseline.server.  
      processors.baselineResource
```

```

        </Resource_Bundle>
        <Resource_Key>40</Resource_Key>
    </Link>
    <Link>
        <Name>BlankSeparatorLine</Name>
        <Action>BlankSeparatorLine</Action>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>
            com.ptc.core.HTMLtemplateutil.server.
            processors.processorsResource
        </Resource_Bundle>
        <Resource_Key>71</Resource_Key>
    </Link>
    <Link>
        <Name>VersionsAndIterations</Name>
        <Action> VersionsAndIterations </Action>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>
            com.MyCompany.CustomResource
        </Resource_Bundle>
        <Resource_Key>28</Resource_Key>
    </Link>
    <Link>
        <Name>LifeCycleHistory_plm</Name>
        <Action>LifeCycleHistory_plm</Action>
        <Tree_Action_Delegate/>
        <In_Minimum_List>true</In_Minimum_List>
        <Resource_Bundle>
            com.ptc.windchill.pdmlink.doc.server.serverResource
        </Resource_Bundle>
        <Resource_Key>20</Resource_Key>
    </Link>
    .
    .
    .

```

6. Create src/com/MyCompany/VersionsAndIterationsNavBarActionDelegate.java and src/com/MyCompany/VersionsAndIterationsNavBarURLActionDelegate.java classes for your action following the instructions in the section of this document called ActionDelegates and URLActionDelegates.

7. Create property file entries so the system can find the ActionDelegate and URLActionDelegate associated with your action. Note that the action name must be in upper case in your property entries:

```
wt.services/svc/default/wt.enterprise.ActionDelegate/VERSIONSANDITERATIONS/wt.doc.WTDocument/0=com.MyCompany.VersionsAndIterationsNavBarActionDelegate/duplicate
```

```
wt.services/svc/default/wt.enterprise.URLActionDelegate/VERSIONSANDITERATIONS/wt.doc.WTDocument/0=com.MyCompany.VersionsAndIterationsNavBarURLActionDelegate/duplicate
```

See the section Properties and Property Files in the chapter Customizing the HTML Client for how to add custom property entries.

8. Create one or more HTML templates for your new page. In this case, you might want to just copy and then modify the existing template for the Versions Details page:

codebase/templates/pdmlink/doc/AllVersionsFrame.html. The modification would consist of adding an IterationHistory table, as shown in the template codebase/templates/pdmlink/doc/IterationHistoryFrame.html.

9. Create a property file entry mapping your new action to the correct HTML template. Assuming the path to your new template is <WT\_HOME>/codebase/templates/MyCompany/VersionsAndIterations.html, this entry would look like:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/VersionsAndIterations/wt.doc.WTDocument/0=templates.MyCompany.VersionsAndIterations
```

See the section Properties and Property Files in the chapter Customizing the HTML Client for how to add custom property entries.

10. Since your new template doesn't need any Windchill script methods other than those in DefaultTemplateProcessor, the latter can be used as your template processor. Create a property file entry to map your new action to this template processor:

```
wt.services/svc/default/wt.enterprise.TemplateProcessor/VersionsAndIterations/wt.doc.WTDocument/0=wt.templateutil.processor.DefaultTemplateProcessor/ duplicate
```

See the section Properties and Property Files in the chapter Customizing the HTML Client for how to add custom property entries.

11. Restart the Method Server. You should now see an All Versions and Iterations item in the navigation bar.

## How to Add a Soft-Type-Specific Details Page Navigation Bar

Like the Actions dropdown menu, the Details page navigation bar is type-based so that the navigation bar for a soft type can be different from that for its parent type. This is done by giving the soft type a different list of navigation links in NavigationAndActions.xml. If no list is specified for the soft type, it will inherit the same set of navigation links as its parent type.

Assume again that you have defined a soft type of WTDocument called Specification Document. Assume also that you want to replace the Versions and Iteration History links with a new All Versions and Iterations link described in the section above in the navigation bar of Specification Documents but not in that of WTDocuments. Instead of modifying the NAVIGATION\_TREE for WTDOCUMENT\_NAVBAR\_ACTIONS you would create a new tree for Specification Document and make your modifications to that tree only as follows:

1. Discover the name of the NAVIGATION\_TREE containing the navigation bar list for the WTDocument Details page. You can locate this information in the file /codebase/typeservices.properties. Search for the string "NAVBAR" and then locate the entry containing "NAVBAR" for wt.doc.WTDocument:

```
wt.services/rsc/default/wt.templateutil.objectPropsNavBar/NAVBAR/  
wt.doc.WTDocument/0=WTDOCUMENT_NAVBAR_ACTIONS
```

2. At the end of this property entry is the name of the tree containing the Actions dropdown list for WTDocuments:  
"WTDOCUMENT\_NAVBAR\_ACTIONS."
3. If you have not already done so, create a customized version of the file <WT\_HOME>/codebase/wt/templateutil/NavigationAndActions.xml as described in the section Actions and Configuration above.
4. Open your custom NavigationAndActions.xml file and copy the NAVIGATION\_TREE named "WTDOCUMENT\_NAVBAR\_ACTIONS." Paste the copy into your file and rename the copy SPECIFICATION\_DOCUMENT\_NAVBAR\_ACTIONS. Your new tree may be placed anywhere in the file as long as it is between existing trees.
5. Delete the links for Versions and Iteration History and add the new link called VersionsAndIterations into the tree for SPECIFICATION\_DOCUMENT\_NAVBAR\_ACTIONS:

```
<NAVIGATION_TREE>  
  <Name>SPECIFICATION_DOCUMENT_NAVBAR_ACTIONS</Name>  
  <Links>  
    <Link>  
      <Name>RelatedParts</Name>  
      <Action>ObjProps</Action>  
      <Tree_Action_Delegate/>  
      <In_Minimum_List>true</In_Minimum_List>  
      <Resource_Bundle>
```

```

        com.ptc.windchill.pdmlink.doc.server.server-
Resource
        </Resource_Bundle>
        <Resource_Key>24</Resource_Key>
</Link>
.
.
.
<Link>
    <Name>VersionsAndIterations</Name>
    <Action> VersionsAndIterations </Action>
    <Tree_Action_Delegate/>
    <In_Minimum_List>true</In_Minimum_List>
    <Resource_Bundle>com.MyCompany.CustomRe
source</Resource_Bundle>
    <Resource_Key>28</Resource_Key>
    <Links/>
</Link>
<Link>
    <Name>LifeCycleHistory_plm</Name>
    <Action>LifeCycleHistory_plm</Action>
    <Tree_Action_Delegate/>
    <In_Minimum_List>true</In_Minimum_List>
    <Resource_Bundle>
        com.ptc.windchill.pdmlink.doc.server.server-
Resource
        </Resource_Bundle>
        <Resource_Key>20</Resource_Key>
</Link>
.
.
.

```

6. Create a typed services property entry mapping your soft type name to your new actions dropdown tree:

```

wt.services/rsc/default/wt.templateutil.objectPropsNavBar/NAVBAR/
WCTYPE|wt.doc.WTDocument|SpecificationDocument/
0= SPECIFICATION_DOCUMENT_NAVBAR_ACTIONS

```

See the section called Properties and Property Files in the chapter Customizing the HTML Resource for more information about adding new property entries.

Complete steps 6-11 described in the section How to Add a Link to a Details Page Navigation Bar above.

## How to Add an Action to the Actions Column of a PDMLink Table

The actions in the actions column of a PDMLinkHTMLTable are specific to the object represented in the table row. Action lists for various object types are contained NAVIGATION\_TREES in the file NavigationAndActions.xml. Property entries map object types to NAVIGATION\_TREE names. Additional properties map an action to an ActionIconDelegate and map an ActionIconDelegate to an icon image file. See the section Tables, Action Columns above for more information.

This section gives an example of how to add an action to the Actions table cell for a given object type. Assume you would like to add an icon for the Update action to tables containing Problem Reports. Currently only the Details action is shown for Problem Reports as follows:



Name	Actions	Number	Change Admin	Priority	Requester	State
pr_2	ⓘ	00361	Administrators	Low	Administrator	Open
pr_1	ⓘ	00362	Administrators	Low	Administrator	Open

To add an Update action to the actions cell for each Problem Report (wt.change2.WTChangeIssue) you would do the following:

1. Discover the NAVIGATION\_TREE in NavigationAndActions.xml containing the table action links for Problem Reports. This can be done by searching the directories in <WT\_HOME>/codebase for .properties files containing the string "iconTableActions." Your search results should look similar to the following:

```
com\ptc\core\foundation\admin\admin.properties(29):
wt.services/rsc/default/wt.templateutil.iconTableActions/
TABLEACTIONS/wt.folder.cabinet/0=CABINET_TABLE_ACTIONS

com\ptc\windchill\cadx\propfiles\HTML.properties(677):
wt.services/rsc/default/wt.templateutil.iconTableActions/
TABLEACTIONS/wt.fc.WTObject/0=WTOBJECT_TABLE_ACTIONS

com\ptc\windchill\pdmlink\doc\doc.properties(209):
wt.services/rsc/default/wt.templateutil.iconTableActions/
TABLEACTIONS/wt.doc.WTDocument/0=WTDOCUMENT_TABLE_ACTIONS

com\ptc\windchill\pdmlink\doc\doc.properties(210):
wt.services/rsc/default/wt.templateutil.iconTableActions/TABLEACTI
ONS/wt.doc.WTDocumentMaster/
0=WTDOCUMENTMASTER_TABLE_ACTIONS
.
.
.
```

Examine the resulting property entries to see if there is an entry for the wt.change2.WTChangeIssue object type. You will find that in the out-of-the-box product there is none. This means that the Problem Report actions are inherited from a superclass. The parent of the wt.change2.WTChangeIssue class is wt.enterprise.Managed. Again, you will find there is no entry for the Managed object type. The parent of the Managed class is wt.fc.WTObject. You will find there is an entry for this class, noted in bold above. The final string in this entry, "WTOBJECT\_TABLE\_ACTIONS" is the name of the tree in NavigationAndActions.xml containing the actions for Problem Reports.

2. Because you only want the Update action to be shown for Problem Reports, and not all WTOBJECTS, you will need to make a new NAVIGATION\_TREE for Problem Report table actions. If you have not already done so, create a customized version of the <WT\_HOME>/codebase/wt/templateutil/NavigationAndActions.xml file as described in the section Actions Configuration earlier in this chapter.
3. In your custom NavigationAndActions.xml file, copy the NAVIGATION\_TREE for WTOBJECT\_TABLE\_ACTIONS. Paste the copy into the file and name the copy "PROBLEM\_REPORT\_TABLE\_ACTIONS." In your new tree, add a <Link> tag for the update action. In this case, the Update action will invoke the same ActionDelegate and URLActionDelegate as the Update action in the dropdown list on the ProblemReport Details page so we can just copy the <LINK> node from the PR\_Details\_Actions NAVIGATION\_TREE. Your new tree should look similar to the following:

```

<NAVIGATION_TREE>
  <Name>PROBLEM_REPORT_TABLE_ACTIONS</Name>
  <Link>
    <Name>Details</Name>
    <Action>ObjProps</Action>
    <Tree_Action_Delegate/>
    <In_Minimum_List>>true</In_Minimum_List>
    <Resource_Bundle>
      com.ptc.core.HTMLtemplateutil.server.
        processors.processorsResource
    </Resource_Bundle>
    <Resource_Key>22</Resource_Key>
    <Links/>
  </Link>
  <Link>
    <Name>Update_PR</Name>
    <Action>Update_PR</Action>
    <Tree_Action_Delegate/>
    <In_Minimum_List>>true</In_Minimum_List>
    <Resource_Bundle>
      com.ptc.windchill.pdmlink.change.server.
        processors.detailsResource
  </Link>
</NAVIGATION_TREE>

```

```

        </Resource_Bundle>
        <Resource_Key>0</Resource_Key>
        <Links/>
    </Link>
</Links>
</NAVIGATION_TREE>

```

In this case, the <Resource\_Bundle> and <Resource\_Key> subtags are used to indicate the rollover text for the icons.

4. Create a property entry mapping the wt.change2.WTChangeIssue object to your new NAVIGATION\_TREE for the wt.templateutil.iconTableActions service:

```
wt.services/rsc/default/wt.templateutil.iconTableActions/TABLEACTIONS/wt.change2.WTChangeIssue/0= PROBLEM_REPORT_TABLE_ACTIONS
```

See the section called Properties and Property Files in the chapter Customizing the HTML Client for more information about adding custom properties.

5. If necessary, create an icon image for your new action and the property entries to map your action to your icon image. In this case, we will use an existing icon: <WT\_HOME>/codebase//com/ptc/core/ui/images/update.gif. This icon is already used for the update action for WTDocuments, as indicated by the following property entries in the file <WT\_HOME>/codebase//com/ptc/windchill/doc/doc.properties:

```
wt.services/svc/default/wt.fc.IconDelegate/UpdateDocument/wt.doc.WTDocument/0=wt.fc.ActionIconDelegate/duplicate
```

```
wt.services/svc/default/wt.fc.ActionIconDelegate/UpdateDocument/wt.doc.WTDocument/0=com.ptc.core.ui.images.update.gif
```

To use these same images for Problem Reports, you can add the following custom properties:

```
wt.services/svc/default/wt.fc.IconDelegate/Update_PR/wt.change2.WTChangeIssue/0=wt.fc.ActionIconDelegate/duplicate
```

```
wt.services/svc/default/wt.fc.ActionIconDelegate/Update_PR/wt.change2.WTChangeIssue/0=com.ptc.core.ui.images.update.gif
```

See the section called Properties and Property Files in the chapter Customizing the HTML Client for more information about adding custom properties.

- Restart your MethodServer. You should now see an Update icon in the Actions column for Problem Reports as shown below:

Name	Actions	Number	Change Admin	Priority	Requester	State
Cracked casing		00122	Administrators	Low	T.H.Hanson	Open
Sensor sensitive to high humidity		00123	Administrators	Medium	B. Carlson	Open

## How to Add a Multiselect Action to the Multiselect Action Bar of a PDMLink Table

PDMLink tables may have a multiselect action bar at the top as shown in the section Tables earlier in this chapter. This section describes how to add an action to an existing multiselect action bar. If you would like to add a multiselect action to a table that does not have an action bar, you will need to modify the table creation code to add a checkbox column and define a multiselect action bar. See the section Tables earlier in this chapter.

It is only possible to add an action to a multiselect action bar for an out-of-the-box client if the table, or at least the action bar for it, is defined on the template using a table service. If the action bar is defined in a java class you will not have access to it. Perform steps 1-2 below to find out if the actions bar is defined on a template.

To illustrate how you would add an action to an existing multiselect action bar, assume you would like to add a link display all the parts related to selected documents in the Related Documents References table. The multiselect action bar of this table currently only contains a link to remove the reference relationship between the context object document and the selected document that references it:



To add a Display Related Parts action to the multiselect action bar you would do the following:

- Discover the NAVIGATION\_TREE in NavigationAndActions.xml containing the multiselect action links for this table. There is no formula for finding the tree --- it will require searching the file for text likely to be contained in the tree name. Often, trees for multiselect action bars have the text "multiselect" in the name so searching on that string will lead you to the

tree. If that fails, search on other likely text --- in this case "references" or "remove" would be likely possibilities.

A search on the text "multiselect" will locate the following tree listing the multiselect actions for this table:

```
<NAVIGATION_TREE>
  <Name>DocumentReferences_MultiSelectActions</Name>
  <Links>
    <Link>
      <Name>DocumentRemoveReference</Name>
      <Action>DocumentRemoveReference</Action>
      <Tree_Action_Delegate/>
      <In_Minimum_List>true</In_Minimum_List>
      <Resource_Bundle>com.ptc.windchill.pdmlink.doc.
        server.processors.processorsResource
      </Resource_Bundle>
      <Resource_Key>91</Resource_Key>
      <Links/>
    </Link>
  </Links>
</NAVIGATION_TREE>
```

2. Try to locate a template referencing this tree name in `<WT_HOME>/codebase/templates/pdmlink`. In this case a search for a file containing the text "DocumentReferences\_MultiSelectActions" will locate the following files:

```
<WT_HOME>/codebase/templates/pdmlink/doc/
DocumentReferencesTableInformation*.html
```

Open the file DocumentReferencesTableInformation.html. You will find the following tableService call within a series of tableService calls defining this table:

```
tableService action=defineActionBar
formName=DocumentReferencesTableInformation
multiSelectActionNavBar=DocumentReferences_MultiSelectAction
s multiSelectActionList=DocumentRemoveReference
actionNavBar=DocumentReferences_TableActions
actionList=DocumentAddReference
```

If your search for files containing the tree name was not successful, you can assume the table of interest is defined in a java class and is not customizable.

3. Once you have found the relevant NAVIGATION\_TREE and template, you are ready to add your new action. Select a unique name for your new action. In this case, we will use "DisplayPartsForReferencesDocs."
4. Create an entry in an .rbInfo file for the label for this action. For example, assume you have a .rbInfo file called

/src/com/MyCompany/CustomResource.rbInfo containing 26 of your custom UI text strings. You would add an entry to that file similar to the following:

```
<27>.value=Display Related Parts  
<27>.constant=DISPLAY_RELATED_PARTS  
<27>.comment=Label for the Display Related Parts action link  
on the Document References table
```

5. Build your .rbInfo file by typing:

```
ResourceBuild com.MyCompany.CustomResource
```

This will generate CustomResource\*.class files in the <WT\_HOME>/codebase/com/MyCompany directory.

6. If you have not already done so, create a customized version of the file <WT\_HOME>/codebase/wt/templateutil/NavigationAndActions.xml as described in the section Action Configuration earlier in this chapter.
7. Add a <LINK> for this action to the DocumentReferences\_MultiSelectActions tree in your custom NavigationAndActions.xml file as follows:

```
<NAVIGATION_TREE>  
  <Name>DocumentReferences_MultiSelectActions</Name>  
  <Links>  
    <Link>  
      <Name>DocumentRemoveReference</Name>  
      .  
      .  
      .  
    <Links/>  
  </Link>  
  <Link>  
    <Name>DisplayPartsForReferencesDocs</Name>  
    <Action> DisplayPartsForReferencesDocs</Action>  
    <Tree_Action_Delegate/>  
    <In_Minimum_List>true</In_Minimum_List>  
    <Resource_Bundle>com.MyCompany.CustomResource  
  </Resource_Bundle>  
    <Resource_Key>27</Resource_Key>  
    <Links/>  
  </Link>  
  
  </Links>  
</NAVIGATION_TREE>
```

The values for <Resource\_Bundle> and <Resource\_Key> should reference the entry you created in steps 4-5. This string will be used for the action bar link and action icon rollover text.

8. Edit the template files DocumentReferencesTableInformation\*.html to include your new action in the actions bar as shown in bold below:

```
tableService action=defineActionBar
formName=DocumentReferencesTableInformation
multiSelectActionNavBar=DocumentReferences_MultiSelectAction
s multiSelectActionList=DocumentRemoveReference,

DisplayPartsForReferencesDocs
actionNavBar=DocumentReferences_TableActions
actionList=DocumentAddReference
```

**Note:** When upgrading PDMLink you will have to merge your customizations into the out-of-the-box template if it has changed in the new release. See the chapter, Windchill Software Maintenance and Best Practices in the Windchill System Administrator's Guide.

9. Create an icon for your new action and map your action name to it by creating the appropriate properties. In this case, our new icon image file is <WT\_HOME>/codebase/com/MyCompany/images/displayRelatedParts.gif, so the property entries would look as follows:

```
wt.services/svc/default/wt.fc.IconDelegate/DisplayPartsForReferenc
esDocs/ wt.doc.WTDocument/0=wt.fc.ActionIconDelegate/duplicate
```

```
wt.services/svc/default/wt.fc.ActionIconDelegate/
DisplayPartsForReferencesDocs/ wt.doc.WTDocument/
0=com.MyCompany.images.displayRelatedParts.gif
```

See the section Properties and Property Files in the chapter Customizing the HTML Client for more information about adding new property entries.

10. You will need a NavBarActionDelegate and NarBarURLActionDelegate for your new action. In this case, no permission checking is needed to determine if the action should be displayed so we will use the DefaultNavBarActionDelegate for the action. Create a new property mapping your action to this delegate:

```
wt.services/svc/default/wt.enterprise.ActionDelegate/
DISPLAYPARTSFORREFERENCEDOCS/java.lang.Object/
0=wt.templateutil.processor.DefaultNavBarActionDelegate/duplicate
```

The URL for this action should call the URLProcessor.URLTemplateAction() method. This method will instantiate the template and template processor for the related parts page and start the processing of the template. To create such a URL, you can use the ActionNavBarURLActionDelegate. (See the section

Multiselect Action Bar earlier in this chapter.) Create a new property mapping the new action to this delegate:

```
wt.services/svc/default/wt.enterprise.URLActionDelegate/  
DISPLAYPARTSFORREFERENCEDOCS/java.lang.Object/  
0=wt.templateutil.processor.ActionNavBarURLActionDelegate/duplicate
```

For the table action bars, the requestor object for the ActionDelegate and URLActionDelegate must be java.lang.Object.

See the section Properties and Property Files in the chapter Customizing the HTML Client for more information about adding new property entries.

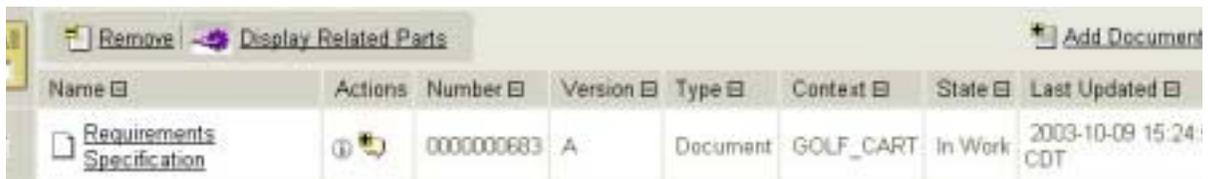
11. You will need to create a template and template processor for the new page to be invoked by this action and then create properties to map your action to these artifacts. If you template is named `<WT_HOME>/codebase/templates/MyCompany/RelatedPartsForReferencesDocs.html` and your template processor is named `<WT_HOME>/codebase/com/MyCompany/RelatedPartsForReferencesDocsProcessor.class` the property entries would be as follows:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/  
DisplayPartsForReferencesDocs /wt.doc.WTDocument/  
0=templates.MyCompany.RelatedPartsForReferencesDocs
```

```
wt.services/svc/default/wt.enterprise.TemplateProcessor/  
DisplayPartsForReferencesDocs/ wt.doc.WTDocument/  
0=com.ptc.MyCompany.RelatedPartsForReferencesDocsProcessor/duplica  
te
```

See the section Multiselect Action Bar earlier in this chapter for how to extract the objects that were checked from the form data in your template processor.

12. Restart the Method Server. You should now see your new link in the actions bar:



## How to Add an Action to the Context-Specific Action Bar of a PDMLink Table

This section describes how to add an action to an existing context-specific table action bar. If you would like to add an action to a table that does not have an action bar, you will need to modify the table creation code to define an action bar. See the section Tables earlier in this chapter.

Adding an action to the context-specific actions bar of a table is very similar to adding an action to the multiselect action bar. As with multiselect actions, it is only possible to add an action for an out-of-the-box client if the table actions bar is defined on the template. If the action bar is defined in a java class you will not have access to it.

Assume that in the example above instead of adding a Display Related Parts action to the multiselect action bar you wanted to add it to the context-specific action bar. Instead of displaying parts related to the selected documents, it will display the parts related to all the documents in the table.

Before modification, the table looks as follows with one action, "Add Document" in the context-specific action bar:



To add the new action you would do the following:

1. Discover the NAVIGATION\_TREE in NavigationAndActions.xml containing the context-specific action links for this table. There is no formula for finding the tree --- it will require searching the file for text likely to be contained in the tree name. Often, trees for context-specific action bars have the text "TableActions" in the name so searching on that string will lead you to the tree. If that fails, search on other likely text --- in this case "reference" or "add" would be likely possibilities.

A search on the text "TableActions" will locate the following tree listing the context-specific actions for this table:

```
<NAVIGATION_TREE>
  <Name>DocumentReferences_TableActions</Name>
  <Links>
    <Link>
      <Name>DocumentAddReference</Name>
      <Action>DocumentAddReference</Action>
      <Tree_Action_Delegate/>
      <In_Minimum_List>true</In_Minimum_List>
    </Link>
  </Links>
</NAVIGATION_TREE>
```

```

        <Resource_Bundle>
            com.ptc.windchill.pdmlink.doc.server.
            processors.processorsResource
        </Resource_Bundle>
        <Resource_Key>92</Resource_Key>
        <Links/>
    </Link>
</Links>
</NAVIGATION_TREE>

```

2. Try to locate a template referencing this tree name in <WT\_HOME>/codebase/templates/pdmlink. In this case a search for a file containing the text "DocumentAddReference" will locate the following files:

```

<WT_HOME>/codebase/templates/pdmlink/doc/DocumentReferencesTableInformation*.html

```

Open the file DocumentReferencesTableInformation.html and search again for this text. You will find the following tableService call within a series of tableService calls defining this table:

```

tableService action=defineActionBar
formName=DocumentReferencesTableInformation
multiSelectActionNavBar=DocumentReferences_MultiSelectActions
multiSelectActionList=DocumentRemoveReference
actionNavBar=DocumentReferences_TableActions
actionList=DocumentAddReference

```

If your search for files containing the tree name was not successful, you can assume the table for interest is defined in a java class and is not customizable.

3. Once you have found the relevant NAVIGATION\_TREE and template, you are ready to add your new action. Select a unique name for your new action. In this case, we will use "DisplayPartsForReferencesDocs" as we did in the previous example.
4. Create an entry in an .rbInfo file for the label for this action. For example, assume you have a .rbInfo file called /src/com/MyCompany/CustomResource.rbInfo containing 26 of your custom UI text strings. You would add an entry to that file similar to the following:

```

<27>.value=Display Related Parts
<27>.constant=DISPLAY_RELATED_PARTS
<27>.comment=Label for the Display Related Parts action link on
the Document References table

```

5. Build your .rbInfo file by typing:

```
ResourceBuild com.MyCompany.CustomResource
```

This will generate CustomResource\*.class files in the <WT\_HOME>/codebase/com/MyCompany directory.

6. If you have not already done so, create a customized version of the file <WT\_HOME>/codebase/wt/templateutil/NavigationAndActions.xml as described in the section Action Configuration earlier in this chapter.
7. Add a <LINK> for this action to the DocumentReferences\_TableActions tree in your custom NavigationAndActions.xml file as follows:

```
<NAVIGATION_TREE>
  <Name>DocumentReferences_TableActions</Name>
  <Links>
    <Link>
      <Name>DocumentAddReference</Name>
      .
      .
      .
    </Link>
    <Link>
      <Name>DisplayPartsForReferencesDocs</Name>
      <Action>DisplayPartsForReferencesDocs</Action>
      <Tree_Action_Delegate/>
      <In_Minimum_List>>true</In_Minimum_List>
      <Resource_Bundle>
        com.MyCompany.CustomResource
      </Resource_Bundle>
      <Resource_Key>2</Resource_Key>
      <Links/>
    </Link>
  </Links>
</NAVIGATION_TREE>
```

The values for <Resource\_Bundle> and <Resource\_Key> should reference the entry you created in steps 4-5. This string will be used for the action bar link and action icon rollover text.

8. Edit the template files DocumentReferencesTableInformation\*.html to include your new action in the actions bar as follows:

```
tableService action=defineActionBar
formName=DocumentReferencesTableInformation
multiSelectActionNavBar=DocumentReferences_MultiSelectActions
multiSelectActionList=DocumentRemoveReference

actionNavBar=DocumentReferences_TableActions
actionList=DocumentAddReference, DisplayPartsForReferencesDocs
```

**Note:** When upgrading PDMLink you will have to merge your customizations into the out-of-the-box template if it has changed in the new release. See the chapter, Windchill Software Maintenance and Best Practices in the Windchill System Administrator's Guide.

9. Create an icon for your new action and map your action name to it by creating the appropriate properties. If your new icon file were `<WT_HOME>/codebase/com/MyCompany/images/displayRelatedParts.gif`, your property entries would look as follows:

```
wt.services/svc/default/wt.fc.IconDelegate/DisplayPartsForReferencesDocs/  
wt.doc.WTDocument/0=wt.fc.ActionIconDelegate/duplicate
```

```
wt.services/svc/default/wt.fc.ActionIconDelegate/ DisplayPartsForReferencesDocs/  
wt.doc.WTDocument/  
0=com.MyCompany.images.displayRelatedParts.gif
```

See the section Properties and Property Files in the chapter Customizing the HTML Client for more information about adding new property entries.

10. You will need a `NavBarActionDelegate` and `NarBarURLActionDelegate` for your new action. In this case, no permission checking is needed to determine if the action should be displayed so we will use the `DefaultNavBarActionDelegate` for the action. Create a new property mapping your action to this delegate:

```
wt.services/svc/default/wt.enterprise.ActionDelegate/  
DISPLAYPARTSFORREFERENCEDOCS/java.lang.Object/  
0=wt.templateutil.processor.DefaultNavBarActionDelegate/duplicate
```

The URL for this action will call the `URLProcessor.URLTemplateAction()` method. This method will instantiate the template and template processor for the related parts page and start the processing of the template. The only information that needs to be passed to the template processor on the URL is the current context object (that is, the document) and the action name. To create such a URL, you can use the `ObjectPropsNavBarURLActionDelegate`. (See the section `Default ActionDelegates` and `URLAction Delegates` earlier in this chapter.) Create a new property mapping the new action to this delegate:

```
wt.services/svc/default/wt.enterprise.URLActionDelegate/
DISPLAYPARTSFORREFERENCEDOCS/java.lang.Object/

0=wt.templateutil.processor.objectpropsNavBarURLActionDelegate/
duplicate
```

See the section `Properties and Property Files` in the chapter `Customizing the HTML Client` for more information about adding new property entries.

11. You will need to create a template for the new page to be invoked by this action and the template processor that will be used for it and then create properties to map your action to these artifacts. If your template is named `<WT_HOME>/codebase/templates/MyCompany/RelatedPartsForReferencesDocs.html` and your template processor is named `<WT_HOME>/codebase/com/MyCompany/RelatedPartsForReferencesDocsProcessor.class` the property entries would be as follows:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
DisplayPartsForReferencesDocs /wt.doc.WTDocument/
0=templates.MyCompany.RelatedPartsForReferencesDocs

wt.services/svc/default/wt.enterprise.TemplateProcessor/
DisplayPartsForReferencesDocs/ wt.doc.WTDocument/
0=com.ptc.MyCompany.RelatedPartsForReferencesDocsProcessor/duplicate
```

Your new template processor might have a Windchill script method, called by the template, to query for all like documents related to the current `ContextObject` and then query for all the parts related to the documents in the list. It could then present the results in a table or other format of your choosing.

12. Restart the Method Server. You should now see your new link in the actions bar:





# 9

## Customizing Search Functionality

This chapter described how to customize various aspects of the search functionality.

<b>Topic</b>	<b>Page</b>
Adding a Search Component.....	9-2
Overriding an Object's Life Cycle State Property .....	9-6
Searching for Soft Types and Attributes .....	9-7
The SearchableAttributes.properties File .....	9-8
Adding Actions to an Object in the Search Results Table .....	9-9
Adding a New Type to the Search User Interface.....	9-10

# Adding a Search Component

## Introduction

The common search component allows a developer to create a “search component” that can be used anywhere within any Windchill product. The `componentId` is used to define the types that can be searched for within the search component, as well as the container types that a search can be constrained within. For example the following entries would create a search component with a `componentId` of “PDMLink.exampleSearch” within the PDMLink solution.

The following search component example will constrain its searches within Products and Libraries and search for Change Issues and Change Requests.

```
PDMLink.exampleSearch.0=wt.change2.WTChangeIssue  
PDMLink.exampleSearch.1=wt.change2.WTChangeRequest2
```

```
PDMLink.exampleSearch.containerType.0=wt.inf.library.WTLibrary  
PDMLink.exampleSearch.containerType.1=wt.pdmlink.PDMLinkProduct
```

The following sections describe how to define a search component, then how to use it.

## Defining a Search Component

Search components can be defined in the following properties file:

```
com\ptc\windchill\enterprise\search\server\SearchableTypes.properties
```

This file already contains search components that can be used as a guide for creating a new one.

The first part of the search component is the Windchill solution with which the component is associated. This part can have one of the following four valid entries:

- Foundation
- WindchillPDM
- PDMLink
- ProjectLink

These four values are linked to the `installed.properties` entries that define the solutions that are installed on your system. If a component is defined with an entry that is not currently installed, it will not work as expected.

The second part of the search component is a unique key that you assign to it. This key is used to determine the object types and container types that will be used within this search component. An example that already exists is the Foundation Container Search functionality.

The Foundation Container Search functionality is an out-of-the-box search component that is used in PDMLink and ProjectLink to search for container type objects. The definition for this search component follows:

```
Foundation.containerSearch.0=wt.projgmt.admin.Project2
Foundation.containerSearch.1=wt.pdmlink.PDMLinkProduct
Foundation.containerSearch.2=wt.inf.library.WTLibrary
```

- The value Foundation indicates that this search component can be used with any Windchill solution that is installed, because Foundation is installed with every Windchill solution.
- The value containerSearch is the key to the component. Each object type is given a unique numerical value as well.

You can also define for a search component the container types to which the search should be constrained. An example that already exists is the sandbox<sup>1</sup> picker constraints that are defined for ProjectLink as follows:

```
# Open Sandbox picker container constraints for use in ProjectLink
ProjectLink.openSandboxConstraint.containerType.0=
  wt.inf.library.WTLibrary
ProjectLink.openSandboxConstraint.containerType.1=
  wt.pdmlink.PDMLinkProduct
```

These definitions are almost identical to the object picker, except for the addition of the containerType keyword to indicate the type of entry.

---

1. The term *sandbox* is used in code and programmer documentation to refer to the functionality that allows you to make PDM data from a PDM environment (that is, Windchill PDM or PDMLink) available to a project, and vice versa.

## Using a Component

Once a component has been defined, it can be used within a picker or as a standalone search client. This can be done by calling either the `CommonSearchPicker.jsp` page or the `CommonSearch.jsp` page. These pages can be configured to set up your search component by passing in the following parameters:

### **appID**

Specifies the solution in which this JSP is to be included, using the solution ID (for example, **Windchill**, **PDMLink**, **ProjectLink**, or **SupplyLink**). URLs and attributes can be displayed differently for different solutions. The default is **Windchill**.

### **selectType**

Specifies the type of selector used in the search results (for example, **multi**, **single**, or **none**). The default is **multi**.

### **componentID**

This parameter has two functions:

- To pass multiple types to search against that will be displayed in the type picker.
- To pass multiple container types to constrain a search (that is, to do a query for all of the objects in a specific set of container types).

The specified types and container types are passed to the search component. The types associated to this attribute are defined the following properties file:

```
com\ptc\windchill\enterprise\search\server\SearchableTypes.properties
```

The `componentId` parameter is not required. If no `componentId` is specified, the aggregate list of searchable types containing a component name of “allSearch” for all installed solutions is returned. The same behavior is true for container types except that, by default, no container types are defined with “allSearch”; this has the net effect of not constraining against any container type.

If an object type is passed for the type picker functionality, that value is used and the `componentId` is ignored.

If a container type and a `componentId` are passed for the container type list functionality, the container type is used and the `componentId` is ignored. Also, for container types, if a combination of object type and `componentId`, but no container type are passed, the container types are resolved using the `componentId`.

### **objectType**

Specifies the object the picker search for (for example, `wt.part.WTPart`). This parameter is not required. If neither `objectType` nor `componentId` are passed, the search uses the aggregate list of all searchable types for all installed solutions with `componentId`'s containing “allSearch” as defined in

SearchableTypes.properties. If a value is passed for this parameter, it overrides the componentId, and the picker searches for only a single type

### **containerRef**

Specifies a container; the search component is then constrained to search within that container.

### **containerType**

Specifies a container type to which a picker is constrained. You can specify a container type in two ways:

- To pass a single container type, use the CONTAINER\_TYPE Search-Objects Webject parameter.
- To pass multiple container types, add entries to the following properties file:

```
com\ptc\windchill\enterprise\search\server\SearchableTypes.properties
```

This allows you to specify multiple container types based on a componentId passed through the Search-Objects Webject. If both a container type and a componentId are passed, the value passed through the CONTAINER\_TYPE parameter overrides the list of container types defined for the componentId. The following is an sample definition:

```
Foundation.allSearch.containerType.0=  
wt.inf.container.ExchangeContainer
```

In this example, Foundation.allSearch is the componentId value, and containerType is required to differentiate between a type and a container type. This allows the same componentId to be used to define a set of types and container types.

## Overriding an Object's Life Cycle State Property

To override the `lifeCycleState` property for a specific object type, add the following XML to that object type's `property.xml` file in the context of that object type:

```
<DataDrivenEnumerator id="dde.lifeCycleState">
  <LabelArea id="la.noSelection" labelType="text" value="">
    <Label resource="-- No Selection --"/>
  </LabelArea>
  <LabelArea id="la.inwork" labelType="text" value="INWORK">
    <Label resource="In Work"/>
  </LabelArea>
  <LabelArea id="la.running" labelType="text" value="RUNNING">
    <Label resource="Running"/>
  </LabelArea>
  <Need attribute="state.state" as="value"/>
  <ModelDoer class="%[:classCoNoOpModelDoer]"/>
</DataDrivenEnumerator>
<ElementGroup id="elementgroup.property.lifeCycleState">
  <LabelArea id="la.lifeCycleState.property.label">
    <Label
resource="[##]com.ptc.windchill.enterprise.config.dca.
enterpriseResource:LIFECYCLESTATE_PTY"/>
  </LabelArea>
  <Layout id="layout.lifeCycleState.ddl">
    <LayoutRow id="layoutRow.lifeCycleState.ddl">
      <Insert ref="dde.lifeCycleState"/>
    </LayoutRow>
    <Option theme="align" param="horizontal" value="left"/>
  </Layout>
</ElementGroup>
```

In the `"dde.lifeCycleState"` element, add or remove `LabelAreas` to add or remove the states you want displayed. The value specified in the `VALUE` attribute of the label area is the value used in the search where clause for the `lifeCycleState` attribute (for example, `state.state='INWORK'`).

## Searching for Soft Types and Attributes

**Note:** This section applies only to Windchill PDMLink and Windchill ProjectLink. This functionality does not apply to Windchill PDM.

The following subsections describe how search capabilities work for site-defined types and attributes created using the Type Manager. For further information about the Type Manager, see the *Windchill Business Administrator's Guide*.

In this discussion, types and attributes created using the Type Manager are called *soft types* and *soft attributes* to distinguish them from site-defined types and attributes that may have been modeled.

### Soft Types in Searches

The first time the object type picker is launched from the Advanced Search page after a soft type has been created, the new type is automatically available for selection. To select the new type, the user must expand the parent type by clicking on the arrow next to the parent type.

When a new soft type is created, there is no need to create additional DCA configuration specifications for the new type because DCA picks up the configuration fragments from the parent types. However, DCA configuration specifications can be added in the context of the new type to override any DCA elements from the parent types.

Soft types can be created at different levels of the organization, at the site and organization levels. The soft types shown depend on where the Advanced Search page was launched and the user who launched it.

### Soft Attributes in Searches

The first time an object type is selected after a soft attribute has been created for it, the new attribute is shown in a table that contains all soft attributes for that object type. By default, the table of soft attributes is hidden. To view them, users must click the **Show More Attributes** link. The **Show More Attributes** link appears only on the search criteria panel for those object types that appear in the Type Manager.

## The SearchableAttributes.properties File

The query layer supports searches against multiple types in a single query for both modeled types and soft types, and their attributes. The means for specifying what attributes and types are to be queried against is the `com.ptc.core.query.common.ResultSpec` class.

If only one type is defined, that type is used as context to specify the `AttributeTypeIdentifiers` for the `ResultSpec` (for example, `WCTYPE|wt.part.WTPart~MBA|masterReference^WCTYPE|wt.part.WTPartMaster~MBA|name`).

If more than one type is defined, `wt.fc.Persistable` is used as the default context for `AttributeTypeIdentifiers` in the `ResultSpec` (for example, `WCTYPE|wt.fc.Persistable~MBA|name`).

Additionally, the query layer expects each `AttributeTypeIdentifier` with a `wt.fc.Persistable` context to have the correct `AttributeTypeIdentifier` type (for example, `ModeledAttributeTypeIdentifier`, `NonPersistedAssociationTypeIdentifier` (for more information, see javadoc for the `com.ptc.core.meta.common` and `com.ptc.core.meta.common.impl` packages). Specifying a context of `wt.fc.Persistable` and an attribute name is not enough information to determine what type of `AttributeTypeIdentifier` to create.

Currently, search functionality against multiple types is the only part of the product that uses this query functionality so the following property file was created to provide a means for determining what type of `AttributeTypeIdentifier` should be created, given a `wt.fc.Persistable` context and an attribute name:

```
com.ptc.windchill.enterprise.search.server.SearchableAttributes.properties
```

You can add entries to `SearchableAttributes.properties` for attributes that should be used in searches against multiple types. Entries are not needed for attributes that will be used in searches against a single type. See the `SearchableAttributes.properties` file for examples of entries and further information.

## Adding Actions to an Object in the Search Results Table

You can customize each object type to display specific actions in the search results table. The configurations for all objects are located in the conf directory; each object type has its own package with its own specific configurations. In the package of every object type is a file named action.xml, which must be modified to add actions. If the action does not currently exist, it must be created. See the section on actions in the *Windchill Client Technology Guide* for details on creating an action.

In the action.xml file is an ActionList with the id actionList.tableRowOperations. This element contains the list of actions that is shown in the search results table. It is important that this ActionList is within the context of the object type. Following is an example:

```
<Context type="<objectType>">

<ActionList id="actionList.tableRowOperations">
  <Insert ref="<action id>"/>
</ActionList>

</Context>
```

To add a new action to an object, you must create the action and then add it to the actionList.tableRowOperations list. Following is an example:

```
<Context type="<objectType>">

<ActionList id="actionList.tableRowOperations">
  <Insert ref="<action id>"/>
  <Insert ref="action.NewAction"/>
</ActionList>

  <Action id="action.NewAction">
    <Extend ref="someAction" />
  </Action>

</Context>
```

There may also be situations where displaying the action is dependent on some condition being set a certain way. You can handle this by using draw handlers or if conditions. Refer to the *Windchill Client Technology Guide* for details on how this is done. Following is an example:

```
<ActionList id="actionList.tableRowOperations" >
  <Insert ref="<action.ConditionalAction>">
<DrawHandler class="<path to handler>">
<Check value="%"<some attribute>]==true"/>
</DrawHandler>
  </Insert>
</ActionList>
```

## Adding a New Type to the Search User Interface

This section describes how to add a new type (either a hard type that has been modeled or a soft created with the Type Manager) to the search user interface using DCA (?) configurations.

First the basic search layout is described, followed by instructions for adding the new type.

### Basic Search Layout

#### Basic Search Configurations

The search component user interface is set up using the following files for each object type.

##### **property.xml**

This file defines all the object-specific properties that are being used to display the search page. Objects inherit all of their parent object type properties as well.

##### **search.xml**

This file defines the layout of the search criteria and search tables for each object type. If no configurations are defined for a specific object type, it inherits its parent's definition.

##### **action.xml**

This file defines the actions that are available to the object type in the search results table.

##### **SearchableTypes.properties**

This file defines the list of object types that are to be used in an all search (?), as well as the types that should be displayed in the object type picker. Out-of-the-box functionality includes all soft types defined from one of the object types listed in this file. It does not, however, include any modeled subclasses.

Any new object type should be defined in this property file to ensure it is displayed in the object type drop-down list so it can be selected from that list.

### The Search Object Model

The search configurations define a number of superclasses and interfaces that are used to provide common functionality for specific object types. The following superclasses and interfaces are defined in the search component:

`wt.enterprise.CabinetManaged`

`wt.enterprise.Managed`

`wt.enterprise.RevisionControlled`

`wt.change2.Changeable2`

wt.workflow.forum.Discussion  
wt.fc.Item  
wt.workflow.notebook.NotebookComponent  
wt.enterprise.Simple  
wt.workflow.engine.WfExecutionObject

## property.xml File

Each object type has a property.xml file that defines the attributes that can be used in the search. These properties are inherited from all superclasses and interfaces that are defined.

## search.xml File

The search.xml file defines the configurations for laying out the object-specific search page to include the search criteria and results tables. The configurations inherit the parent's class configurations if none are specified for the object type.

Following are the elements to use for specific purposes in the search.xml file.

- `<LayoutRow id="layoutRow.nameNumber">`  
This LayoutRow is used to define an object's identifying attributes, such as name or number, directly below the keyword field, if one exists.
- `<ElementGroup id="elementGroup.objectSpecificAttributes">`  
This ElementGroup defines the object attributes that are displayed on the search page below the object type picker. Usually, they do not contain the object's identifying attributes, such as name or number.
- `<ElementGroup id="elementGroup.layoutRow.softAttributes" />`  
This ElementGroup is defined for Typeable objects to allow the search page to render the soft attributes that are defined for an object type. If you do not want to show the "More Attributes Link", include this line to override it.
- `<CompositeTable id="compositeTable.searchResults">`  
This table defines the search results table for each object type. It can be overridden or extended, depending on the new objects attributes that you want to display.
- `<ElementGroup id=":  
:elementgroup.nameAndImage.compositeTableColumns">`  
This column is used to define the name and image column for an object type.

## action.xml File

This file defines the object-specific actions, such as details page, name link, and so on. See Adding Actions to an Object in the Search Results Table for more information.

## Adding a New Type

All soft types are automatically added to the object type picker as a subclass of the modeled type from which they were created. When a user selects this object, the parent class configuration files are used. The "More Attributes Link" can be used to search against all of the soft attributes defined for the new soft type.

If a new modeled (hard) type object is to be added to the search page, and the object is a subclass of an existing search type, you can add that new type to SearchableTypes.properties and the object will then appear in the object type picker. Additionally, all searches performed on the parent class will include the subtypes.

If the new hard type is not a subclass of an existing search type, or you want to define different configurations for this new type, add the following files into your conf directory:

- property.xml
- search.xml
- action.xml

## property.xml File

In this file, define the properties that are specific to the new object type. All properties already defined in parent classes can be used by the new type as well.

Following is an example property file that defines the attributes for Projects. Characters shown in bold type are of particular relevance to this discussion.

```
<?xml version="1.0" standalone="no"?>
<!--Define the repository package that this class belongs, this should be the
directory under conf-->
<Repository package="com.ptc.windchill.projectlink.project2">

<!--Import any packages in which you want to re-use other code-->
  <Import package="com.ptc.windchill.enterprise"/>
  <Import package = "com.ptc.windchill"/>

<!--Set up the context for the the Synonym, in this case "Project2". This will be
used in other files to define the context for this object. The value must equal the
actual type name of the object ->

  <Context>
    <Synonym id="Project2" value="wt.projmgmt.admin.Project2"/>
  </Context>

<!--Define the properties within the context of the synonym defined above. The id
```

is used to define the property so that it can be referenced in other locations, i.e. search criteria/search results tables. The Extend is the property that you want to extend. You can extend an existing property, or a class of some kind. The need attribute is the actual attribute that will be used to gather the information from Windchill. The Label is a string that defines the label on the page. These Labels are all localized into Resource Bundles.-->

```

<Context type="%[Project2]">
  <!-- Properties -->
  <Property id="property.name">
    <Extend ref=":property.string"/>
    <Need attribute="containerInfo.name"/>
    <Label
resource=" [##]com.ptc.netmarkets.project.projectResource:PROJECT2_NAME_PTY" />
  </Property>
  <Property id="property.scope">
    <Extend ref=":property.string" />
    <Need attribute="scope"/>
    <Label
resource=" [##]com.ptc.windchill.enterprise.Item.client.itemResource:SCOPE_PTY" />
  </Property>
  <Property id="property.businessLocation">
    <Extend ref=":property.string" />
    <Need attribute="businessLocation"/>
    <Label
resource=" [##]com.ptc.netmarkets.project.projectResource:BUSINESS_LOCATION_PTY" />
  </Property>
  <Property id="property.businessUnit">
    <Extend ref=":property.string" />
    <Need attribute="businessUnit"/>
    <Label
resource=" [##]com.ptc.netmarkets.project.projectResource:BUSINESS_UNIT_PTY" />
  </Property>
  <Property id="property.riskDescription">
    <Extend ref=":property.string" />
    <Need attribute="riskDescription" />
    <Label
resource=" [##]com.ptc.windchill.enterprise.Item.client.itemResource:RISKDESCRIPTION
_PTY" />
  </Property>
  <Property id="property.statusDescription">
    <Extend ref=":property.string" />
    <Need attribute="statusDescription" />
    <Label
resource=" [##]com.ptc.windchill.enterprise.Item.client.itemResource:STATUSDESCRIPTI
ON_PTY" />
  </Property>
  <Property id="property.containerInfo.description">
    <Extend ref=":property.string" />
    <Need attribute="containerInfo.description"/>
    <Label
resource=" [##]com.ptc.windchill.enterprise.Item.client.itemResource:DESCRIPTION_PTY
"/>
  </Property>
</Context>
</Repository>

```

## search.xml File

Following are the elements to use for specific purposes in the search.xml file.

- To define the name and number row which other objects have, use the following element:

```
<LayoutRow id="layoutRow.nameNumber">
```

This LayoutRow is used to define an object's identifying attributes, such as name or number, directly below the keyword field, if one exists.

- To add additional criteria, add the following element group:

```
<ElementGroup id="elementGroup.objectSpecificAttributes">
```

This ElementGroup defines the object attributes that are displayed on the search page below the object type picker. Usually, they do not contain the object's identifying attributes, such as name or number.

If the object is typeable and you want to define soft attributes, do not include this element group. If you do not want to show the "More Attributes Link", include the following line to override it:

```
<ElementGroup id="elementGroup.layoutRow.softAttributes" />
```

- To define the search results that you want to display for this object type, add the following element:

```
<CompositeTable id="compositeTable.searchResults">
```

```
<ElementGroup id="
:elementgroup.nameAndImage.compositeTableColumns">
```

This column is used to define the name and image column for an object type.

Following is an example search.xml file that was defined for Windchill ProjectLink. Characters shown in bold type are of particular relevance to this discussion.

```
-- <?xml version="1.0" standalone="no"?>

<!--Define the package, again this is the directory under conf-->
<Repository package="com.ptc.windchill.projectlink.project2">

<!--Import any packages in which you want to re-use other code-->
  <Import package="com.ptc.core.foundation.persistable"/>
  <Import package="com.ptc.windchill.enterprise.search"/>
<Import package="com.ptc.windchill"/>
<!--Set up the context type to be used, in this case the Synonym from property.xml-->
  <Context type="%[Project2]">

    <!-- Search Criteria -->
<!--This ElementGroup defines the attributes that are specific to Project2. The
properties referenced here are the properties defined in property.xml or some other
parent class. The sticky value sets the sticky, whether or not DCA will remember
your last value. Out of the box, this is set to false so that outside DCA and Saved
```

**Searches can be used to populate attributes->**

```
<ElementGroup id="elementGroup.objectSpecificAttributes">
<Extend ref="persistable:elementGroup.objectSpecificAttributes"/>
<!-- Empty Spacer Row -->
<Insert ref=":layoutRow.spacer"/>
<!--Each LayoutRow must have a unique ID within this context-->
<LayoutRow id="rowAttr1">
<Insert ref=":property.businessLocation" sticky="%[context#sticky]"/>
<Insert ref=":property.businessUnit" sticky="%[context#sticky]"/>
</LayoutRow>
<!-- Empty Spacer Row -->
<Insert ref=":layoutRow.spacer"/>
<LayoutRow id="rowAttr2">
<Insert ref=":property.containerInfo.description"
sticky="%[context#sticky]"/>
<Insert ref=":property.riskDescription"/>
</LayoutRow>

<!-- Empty Spacer Row -->
<Insert ref=":layoutRow.spacer"/>
<LayoutRow id="rowAttr3">
<Insert ref=":property.statusDescription" sticky="%[context#sticky]"/>
<Insert ref=":elementgroup.picker.organization"
if="%[property#wt.properties#wt.org.OrganizationOwned.displayOrganization]==true"/>
</LayoutRow>
</ElementGroup>
```

**<!-- We don't want to display the "Show More Attributes" link for this type, so we're overriding the default with an empty elementgroup -->**

```
<ElementGroup id="elementGroup.layoutRow.softAttributes" />
```

**<!--This layout row defines the object identity items that you see at the top of the page, in most cases it is name and number, but in this case its name and scope-->**

```
<!-- Layout Rows -->
<LayoutRow id="layoutRow.nameNumber">
<Insert ref=":property.name" sticky="%[context#sticky]"/>
<Insert ref=":property.scope" sticky="%[context#sticky]"/>
</LayoutRow>
```

**<!--The table below defines the search results table for this object type.--**

```
<!-- Tables -->
<!-- Search Results table for this object type -->
<CompositeTable id="compositeTable.searchResults">
<Extend ref="search:compositeTable.searchResults.base"/>
<Insert ref=":elementgroup.nameAndImage.compositeTableColumns"/>
<CompositeColumn show=":actionList.tableRowOperations"
if="%[context#isPicker]!=true">
<Label
resource="[#]com.ptc.core.foundation.persistable.client.persistableResource:ACTION
S_LBL"/>
</CompositeColumn>
<CompositeColumn show=":property.containerName">
<Label
resource="[#]com.ptc.core.foundation.persistable.client.persistableResource:CONTAI
NER_LBL"/>
```

```

        <Insert ref=":layoutRow.sortingActions"/>
    </CompositeColumn>
    <CompositeColumn show=":property.updatestamp">
        <Label
resource="[[#]com.ptc.core.foundation.persistable.client.persistableResource:UPDATE
STAMP_LBL"/>
        <Insert ref=":layoutRow.sortingActions"/>
    </CompositeColumn>
    <CompositeColumn show=":property.picker.organization"
if="%[property#wt.properties#wt.org.OrganizationOwned.displayOrganization]==true">
        <Label
resource="[[#]com.ptc.core.foundation.principal.group.organization.client.wtorgReso
urce:ORGANIZATION_NAME_LBL"/>
        <Insert ref=":layoutRow.sortingActions"/>
    </CompositeColumn>
    </CompositeTable>
    </Context>
</Repository>

```

## action.xml File

This file defines the actions in the actions column, as well as the actions associated with the name link for an object. See customizing actions for a detailed description of this file.

```

<?xml version="1.0" standalone="no"?>
<Repository package="com.ptc.windchill.projectlink.project2">

    <Import package="com.ptc.windchill.enterprise" />
    <Context type="%[Project2]">

        <ActionList id="actionList.tableRowOperations"
actionListType="horizontal">
            <Insert ref="hyperlink.view.detailPage"/>

        </ActionList>

        <HyperLink id="hyperlink.view.detailPage" labelType="imageText" drawType="image"
window="top">
            <Image resource="/wtcore/images/com/ptc/core/ca/web/misc/details.gif"/>
            <ToolTip
resource="[[#]com.ptc.netmarkets.project.projectResource:PROJECT2_TOOL_TIP"/>
            <!-- Create a ProjectLink URL if the object was created in PJJ -->
            <HRef resource="/netmarkets/jsp/project/details.jsp"/>
            <!-- Create a ProjectLink URL if the object was created in PJJ -->
            <ModelHandler
class="com.ptc.core.ca.co.client.windchill.CoSimpleWindchillHyperLinkProducer"/>
        </HyperLink>

        <HyperLink id="hyperlink.nameLink" labelType="imageText" drawType="text"
window="top">
            <Label resource="{0}">
                <Need attribute="containerInfo.name" />
            </Label>
            <!-- Create a ProjectLink URL if the object was created in PJJ -->
            <HRef resource="/netmarkets/jsp/project/details.jsp"/>
            <!-- Create a ProjectLink URL if the object was created in PJJ -->

```

```
    <ModelHandler  
class="com.ptc.core.ca.co.client.windchill.CoSimpleWindchillHyperLinkProducer" />  
    </HyperLink>  
  
    </Context>  
</Repository>
```



# 10

## Customizing Change Management

<b>Topic</b>	<b>Page</b>
Overview .....	10-2
Prerequisite Knowledge .....	10-2
Change Objects and Their Relationships .....	10-3
HTML Hidden Input Form Fields .....	10-4
Expand and Collapse Functionality .....	10-4
Generating the Change Management User Interface .....	10-6
Changing the Look of the User Interface .....	10-18
Processor/Template Reference Tables .....	10-31
Adding an Attribute to WTChangeRequest2 .....	10-34
Change Management Delegates .....	10-43

## Overview

This chapter describes how the change management user interface is generated and gives examples of how to customize the user interface to fit your requirements. (Chapter 9, Customizing Solutions, also describes how to customize change management workflow process templates.)

The following general topics are included:

- Prerequisite knowledge that you are assumed to have.
- A brief description of each of the change objects and how they work in relationship to one another.
- A description of the use of HTML hidden input form fields to help control an object's mode (create, update, or view), as well as an object's expand state (expanded or collapsed). Throughout this chapter, expand state refers to whether an object is expanded or collapsed and should not be confused with life cycle state.
- A description of the template processors and HTML templates that are used to generate the change management user interface and how they work together. Although the user interface is rendered as one page, many subtemplates are used to generate the output. The form task delegates that perform user actions in the user interface are also described.
- An explanation of how to make changes to the look of the user interface and some examples.
- An example of how to add an attribute to a change request.
- A description of the change management delegates.

## Prerequisite Knowledge

These topics and examples assume you are familiar with standard HTML, template processing, subtemplate processing, table service, task delegates, and context properties (object, action, and form data). For further information on these topics, see Chapter 7, Customizing the HTML Client.

You must be familiar with the following classes:

- `wt.templateutil.processor.GenerateFormProcessor`
- `wt.enterprise.BasicTemplateProcessor`
- `wt.templateutil.processor.FormTaskDelegate`
- `wt.templateutil.table.BasicTableService`
- `wt.templateutil.processor.SubTemplateService`

# Change Objects and Their Relationships

Change management uses seven PDM objects: change issue, change request, change investigation, change proposal, analysis activity, change order, and change activity. Following are brief descriptions of their use and their relationships with each other.

## **Change issue**

Stores a comment, suggestion, or problem that relates to some PDM-managed information. For example, a change issue might describe a defect in a part. A change issue can originate from any source, internal (an employee) or external (a customer). If a change issue is deemed important or worthy of further investigation, the change process is started by the creation of a change request.

## **Change request**

Begins a formal change process. It signifies a request from a PDM user to formally investigate a problem. A change request serves as the organizing object for a group of related change objects. That is, a user can find all the information regarding a given change process through its change request.

## **Change investigation**

Holds the cause of the root problem for a given change request. For a very complex problem, the change investigation will probably relate to one or more analysis activities that contain detailed information.

## **Change proposal**

Holds a proposed solution for a given change request. For a complex problem, the change proposal will probably relate to one or more analysis activities that contain detailed information.

## **Analysis activity**

Holds detailed analysis results for either a change investigation or a change proposal. These analysis results assist in determining the cause of the root problem (in a change investigation) or support the proposal to fix the problem (in a change proposal).

## **Change order**

Gives authority either to make a change to existing changeables,<sup>1</sup> or to introduce new changeables for the purpose of addressing a given change request.

## **Change activity**

Represents a work instruction that must be completed to satisfy a given change order. A complex change order may have many change activities, while a simple change order may have only a single change activity.

---

1. At this time, only parts and documents are considered changeables.

## HTML Hidden Input Form Fields

Several objects may be displayed on a single page of the change management user interface. HTML hidden form fields are used to determine the following aspects of these objects:

- Which of the objects are expanded
- Which of the objects (if any) should be in update mode
- If a new object is to be created

At any given time, any number of objects can be expanded, but only one object can be in create or update mode.

The remainder of this section describes how HTML hidden form fields provide the functionality to expand and collapse objects, and create or update objects.

## Expand and Collapse Functionality

The change management form allows users to expand and collapse objects to control how much information they want to view at one time. Expand and collapse functionality does not apply to the change request. Because the change request is considered the organizing object for the change process, it is expanded at all times. However, all other objects can be expanded and collapsed.

The expanded state of each object is tracked using hidden form fields. To indicate that an object is expanded, the structure of a hidden form field is similar to the following:

```
<INPUT Type=hidden NAME="OR:wt.change2.WTChangeOrder2:2371479"  
      VALUE="expand" >
```

The name of the hidden form field reflects the object ID and the value indicates that it should be expanded. During processing of the form data, when the object with the ID `wt.change2.WTChangeOrder2:2371479` is set as the context object, the processor knows by the form data property and value pair that the full detail of the object must be displayed. An object that is collapsed does not have an associated hidden form field.

## Create and Update Functionality

The hidden form fields named `create_class` and `create_parent` are used to control the insertion of a blank, updateable section into the page for the purpose of creating a new object. Creation of an object is triggered by setting the value of the hidden form field `create_class` to the class of object that will be created. When the page is loaded initially, this field is empty. If the user selects a New <object> link, a Javascript method sets the value of `create_class` to a string representation of the class of the new object. A complimentary hidden form field, `create_parent`, is also set to the value of the parent ID for the new object. For example, if the object to create is a change investigation, the parent ID should be the ID of the change request. When the Investigate section is being generated, the following script call triggers the method `displayCreateNewObject` in the `ChangeInvestigationProcessor`:

```
<SCRIPT LANGUAGE=Windchill>
<!--
displayCreateNewObject class=wt.change2.WTChangeInvestigation
-->
</SCRIPT>
```

The `displayCreateNewObject` method evaluates the `create_class` form field, finds that a new change investigation must be created, and inserts the create form using the `CreateChangeInvestigation.html` template. The hidden form fields are added to the generated code, as follows, when the form is inserted into the user interface:

```
<INPUT Type=hidden NAME="create_parent"
VALUE="OR:wt.change2.WTChangeRequest2:2344567">
<INPUT Type=hidden NAME="create_class"
VALUE="wt.change2.WTChangeInvestigation">
```

When a user clicks the Accept link, these form fields are used to determine the type of the new object and its parent.

The hidden form field named `update` is used to indicate which object is to be presented in update mode. When the page is loaded initially, the value for this field is empty. If the user selects an Update link for an object, Javascript (found in `DefaultChangeTask.html`) is used to set the value of the update form field equal to the object ID, as shown in the following example:

```
<INPUT Type=hidden NAME="update"
VALUE="OR:wt.change2.WTChangeActivity2:2371601">
```

The existence of this hidden form field instructs the template processor to render the object so that the user is able to update its attributes. The processor knows by the `update` hidden form field that the update template, rather than a read-only view template, should be used to output the form fields for the object. When the page is regenerated with update mode, this hidden form field is added to the source with the value equal to the object ID.

# Generating the Change Management User Interface

This section describes how the change management form (page) is generated by template processors and how user actions (create, update, and delete) are performed by form task delegates.

## Template Processors

The change management user interface is made possible using template processing, subtemplate processing, and the table service. (For further information on these topics, see chapter 7, Customizing the HTML Client.) The change request is considered the organizing object which leads the generation of the page.

The remainder of this section describes the template processors used, in combination with various templates, to generate specific aspects and parts of the change management page:

- **ChangeManagementFormProcessor**, which initiates generation of the page and ensures the proper change request is set as the context object.
- **DefaultChangeTaskProcessor**, which helps control which sections of the page (Request, Investigate, Propose, and Implement) are generated and the order in which they are presented on the page.
- **ChangeRequestProcessor**, which helps generate the Request section of the page.
- **ChangeInvestigationProcessor**, which helps generate the Investigate section of the page.
- **ChangeProposalProcessor**, which helps generate the Propose section of the page.
- **ChangeOrderProcessor**, which helps generate the Implement section of the page.
- **AnalysisActivityProcessor**, which helps generate the expanded view of an analysis activity.
- **ChangeActivityProcessor**, which helps generate the expanded view of a change activity.

## ChangeManagementFormProcessor

Generation of the change management form can be triggered from a local search, the initial Change Manager application page, the Windchill Explorer, or a workflow task.

### Triggering from a Local Search, the Change Manager Page, or the Windchill Explorer

The starting points for generation of the change form are the template processor `ChangeManagementFormProcessor`, in combination with the `ChangeManagementForm.html` template. Selecting any change object can trigger the generation of the change form, but a change request always leads the generation of the page. Therefore, the first step performed by `ChangeManagementFormProcessor` is to determine if the context object is an instance of a change request. If it is not, the change request for the change object is found and set as the context object.

This processor and template combination is the shell for the entire change form. Windchill script calls within the `ChangeManagementForm.html` template produce output that displays the page title, adds the form tag (name, action, method), and triggers the next processor and template combination. Also included in `ChangeManagementForm.html` is the Javascript used when loading and unloading the page.

The object-specific information visible to the user (for example, change request attributes, a list of change proposals, and so on) is generated using subtemplates. A Windchill script call in the `ChangeManagementForm.html` template triggers the method `displayChangeTask` (which is defined in `ChangeManagementFormProcessor`). The `displayChangeTask` method immediately invokes `DefaultChangeTaskProcessor`, which determines which sections are generated and the order in which they are presented on the page (as described later in this section).

### Triggering Through a Workflow Task

You can also trigger the display of a change object through the Worklist. The Worklist is a table of all workflow tasks. From the column labeled Primary Business Object, you can click on the link associated with a change object and the same process described in the preceding subsection is followed.

From the task list, the change management form is embedded in the workflow task. For example, if the task is to Implement the Change, you can select the link from the Task column. This loads the task detail page with instructions to complete the task and the change management form embedded.

## DefaultChangeTaskProcessor

DefaultChangeTaskProcessor works in combination with the DefaultChangeTask.html template to control which sections are generated and in which order. By default, the four main sections are Request, Investigate, Propose, and Implement. The DefaultChangeTask.html template contains four Windchill script calls, one for each section. These script calls trigger the generation of a particular section. For example, the following script call triggers the generation of the Investigate section:

```
<Script language=Windchill>
  displayChangeInvestigationSection
</Script>
```

By removing this script call, you prevent the entire Investigate section from being generated. However, the other sections -- Request, Propose, and Implement -- are not affected and still appear.

The four script calls in DefaultChangeTask.html invoke four different processors: ChangeRequestProcessor, ChangeInvestigationProcessor, ChangeProposalProcessor, and ChangeOrderProcessor. The following figure illustrates the four sections and the processor/template combination that is used to generate each.

Note that DefaultChangeTaskProcessor and DefaultChangeTask.html are not referenced in the figure. Although they contribute to the generation of the user interface, they do not generate code that is visible to the user. Instead, they are responsible for the order in which the sections are displayed; they initialize the hidden form fields (create\_parent, create\_class, and update); and they contain client-side Javascript.

The screenshot shows the Windchill Change Manager interface in Microsoft Internet Explorer. The page is titled "CHANGE MANAGER" and displays four main sections: Request, Investigate, Propose, and Implement. Each section is associated with a specific processor and template, as indicated by the text on the right side of the image.

Section	Processor/Template
Request	Generated by ChangeManagementFormProcessor/ ChangeManagementForm.html
Investigate	Generated by ChangeInvestigationProcessor/ ChangeInvestigationSection.html
Propose	Generated by ChangeProposalProcessor/ ChangeProposalSection.html
Implement	Generated by ChangeOrderProcessor/ ChangeOrderSection.html

## ChangeRequestProcessor

ChangeRequestProcessor is invoked from the DefaultChangeTaskProcessor in a method named displayChangeRequestSection. Several HTML templates are used in combination with this processor. These templates are used to display the Request section header, as well as detailed information about the change request.

The following three templates are used in conjunction with this processor:

- ViewChangeRequest.html
- UpdateChangeRequest.html
- CreateChangeRequest.html

The following figure shows a change request in view mode. The template ViewChangeRequest.html is used to generate this view.

Generated by ChangeManagementFormProcessor/  
ChangeManagementForm.html

Generated by ChangeRequestProcessor/  
ViewChangeRequest.html

Generated by ChangeInvestigationProcessor/  
ChangeInvestigationSection.html

Generated by ChangeProposalProcessor/  
ChangeProposalSection.html

Generated by ChangeOrderProcessor/  
ChangeOrderSection.html

## ChangeInvestigationProcessor

The Windchill script call displayChangeInvestigationSection from the DefaultChangeTask.html template invokes the method displayChangeInvestigationSection in DefaultChangeTaskProcessor. This method sets the context action to ChangeInvestigationSection and then calls a subtemplate service to find the appropriate processor for that action: ChangeInvestigationProcessor.

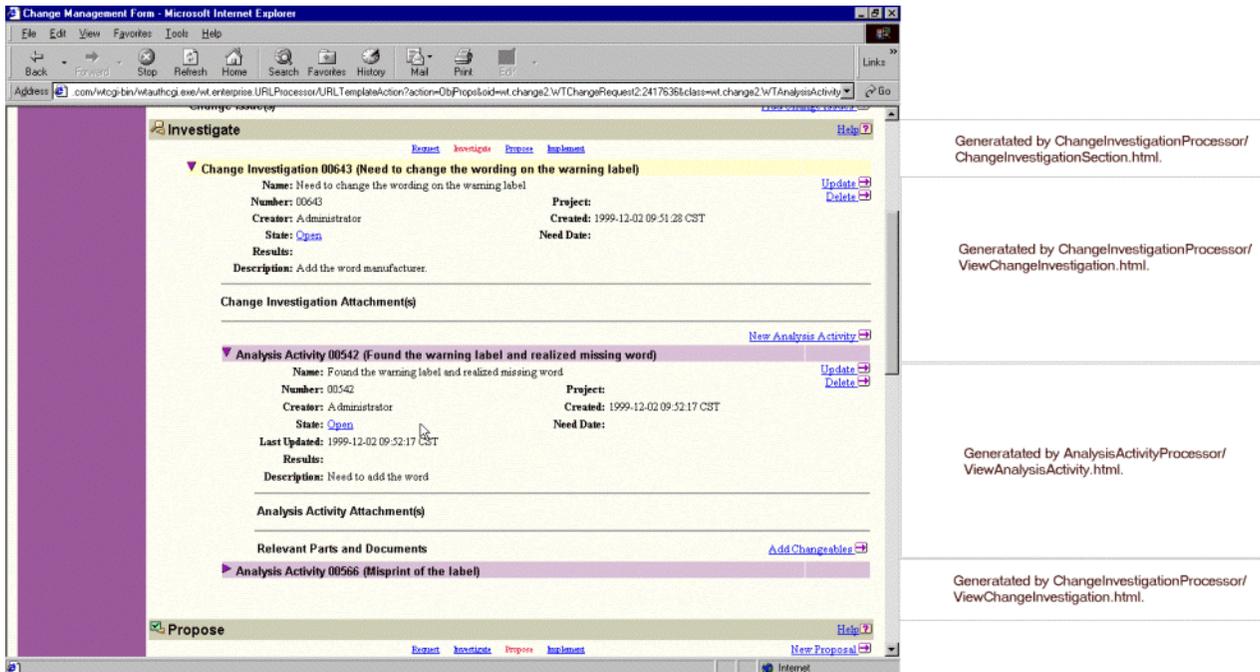
ChangeInvestigationProcessor, in combination with the ChangeInvestigationSection.html template, generates the Investigate section of the page. This section includes the header line, the New Change Investigation link, and the list of change investigations for the change request.

The method `displayChangeInvestigationTable` is used to query for all the change investigations and write a header line for each row returned. If the current row is to be in expanded form, the context action (`ViewChangeObject` or `UpdateChangeObject`) is changed according to the mode for that object (using the form data, as explained earlier in this chapter under HTML Hidden Input Form Fields, to determine if the object is in update mode).

Given either of the actions `ViewChangeObject` or `UpdateChangeObject`, the subtemplate service is used to find the appropriate template -- either `ViewChangeInvestigation.html` or `UpdateChangeInvestigation.html` -- and the same processor, `ChangeInvestigationProcessor`. These files display detail about the change investigation, as well as the links to trigger an action to be performed. These actions include `Update` and `Delete` if in view mode, and `Accept` and `Cancel` if in update mode. Also included in view mode is the link `New Analysis Activity`. This link inserts a form to create an analysis activity that will be associated with the change investigation (see `AnalysisActivityProcessor` later in this section).

After all the existing change investigations have been processed, the last script call in the `ChangeInvestigationSection.html` template is used to determine if a new investigation is to be created. A form field named `create_class` is used to store the class to be created. If the value matches the class `WTChangeInvestigation`, the context action is changed to `CreateChangeObject` and the subtemplate service finds the appropriate processor and template: `ChangeInvestigationProcessor` and `CreateChangeInvestigation.html`.

The following figure shows the Investigate section in view mode with the change investigation and one analysis activity expanded.



## ChangeProposalProcessor

The Windchill script call `displayChangeProposalSection` from the `DefaultChangeTask.html` template invokes the method `displayChangeProposalSection` in `DefaultChangeTaskProcessor`. This method sets the context action to `ChangeProposalSection` and then calls a subtemplate service to find the appropriate processor for that action: `ChangeProposalProcessor`.

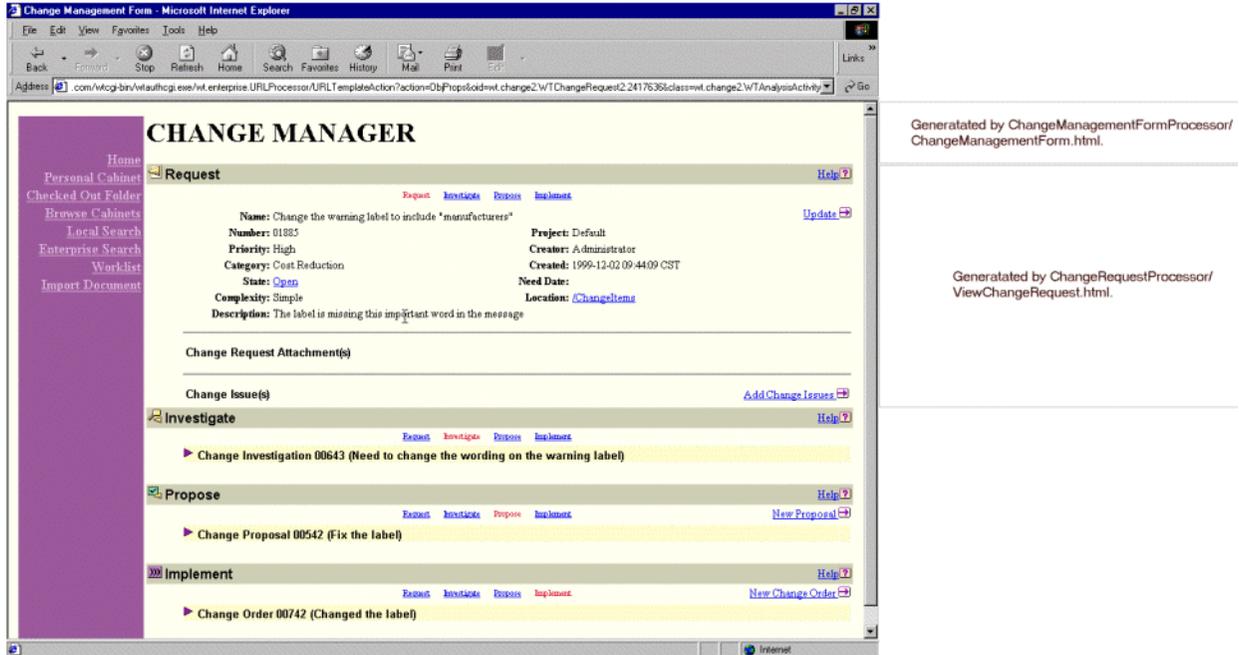
`ChangeProposalProcessor`, in combination with the `ChangeProposalSection.html` template, generates the Propose section of the page. This section includes the header line, the New Change Proposal link, and the list of change proposals for the change request.

The method `displayChangeProposalTable` is used to query for all the change proposals and write a header line for each row returned. If the current row is to be in expanded form, the context action (`ViewChangeObject` or `UpdateChangeObject`) is changed according to the mode for that object (using the form data, as explained earlier in this chapter under HTML Hidden Input Form Fields, to determine if the object is in update mode).

Given either of the actions `ViewChangeObject` or `UpdateChangeObject`, the subtemplate service is used to find the appropriate template -- either `ViewChangeProposal.html` or `UpdateChangeProposal.html` -- and the same processor, `ChangeProposalProcessor`. These files display detail about the change proposal, as well as the links to trigger an action to be performed. These actions include Update and Delete if in view mode, and Accept and Cancel if in update mode. Also included in view mode is the link New Analysis Activity. This link inserts a form to create an analysis activity that will be associated with the change proposal (see `AnalysisActivityProcessor` later in this section).

After all the existing change proposals have been processed, the last script call in the `ChangeProposalSection.html` template is used to determine if a new proposal is to be created. A form field named `create_class` is used to store the class to be created. If the value matches the class `WTChangeProposal`, the context action is changed to `CreateChangeObject` and the subtemplate service finds the appropriate processor and template: `ChangeProposalProcessor` and `CreateChangeProposal.html`.

The following figure shows the Propose section with a change proposal expanded in view mode and the form for a new proposal in create mode.



## ChangeOrderProcessor

The Windchill script call `displayChangeOrderSection` from the `DefaultChangeTask.html` template invokes the method `displayChangeOrderSection` in `DefaultChangeTaskProcessor`. This method sets the context action to `ChangeOrderSection` and then calls a subtemplate service to find the appropriate processor for that action: `ChangeOrderProcessor`.

`ChangeOrderProcessor`, in combination with the `ChangeOrderSection.html` template, generates the `Implement` section of the page. This section includes the header line, the `New Change Order` link, and the list of change orders for the change request.

The method `displayChangeOrderTable` is used to query for all the change orders and write a header line for each row returned. If the current row is to be in expanded form, the context action (`ViewChangeObject` or `UpdateChangeObject`) is changed according to the mode for that object (using the form data, as explained earlier in this chapter under `HTML Hidden Input Form Fields`, to determine if the object is in update mode).

Given either of the actions ViewChangeObject or UpdateChangeObject, the subtemplate service is used to find the appropriate template -- either ViewChangeOrder.html or UpdateChangeOrder.html -- and the same processor, ChangeOrderProcessor. These files display detail about the change order, as well as the links to trigger an action to be performed. These actions include Update and Delete if in view mode, and Accept and Cancel if in update mode. Also included in view mode is the link New Change Activity. This link inserts a form to create a change activity that will be associated with the change order (see ChangeActivityProcessor later in this section).

After all the existing change orders have been processed, the last script call in the ChangeOrderSection.html template is used to determine if a new order is to be created. A form field named create\_class is used to store the class to be created. If the value matches the class WTChangeOrder2, the context action is changed to CreateChangeObject and the subtemplate service finds the appropriate processor and template: ChangeOrderProcessor and CreateChangeOrder.html.

The following figure shows the Implement section with a change order in update mode.

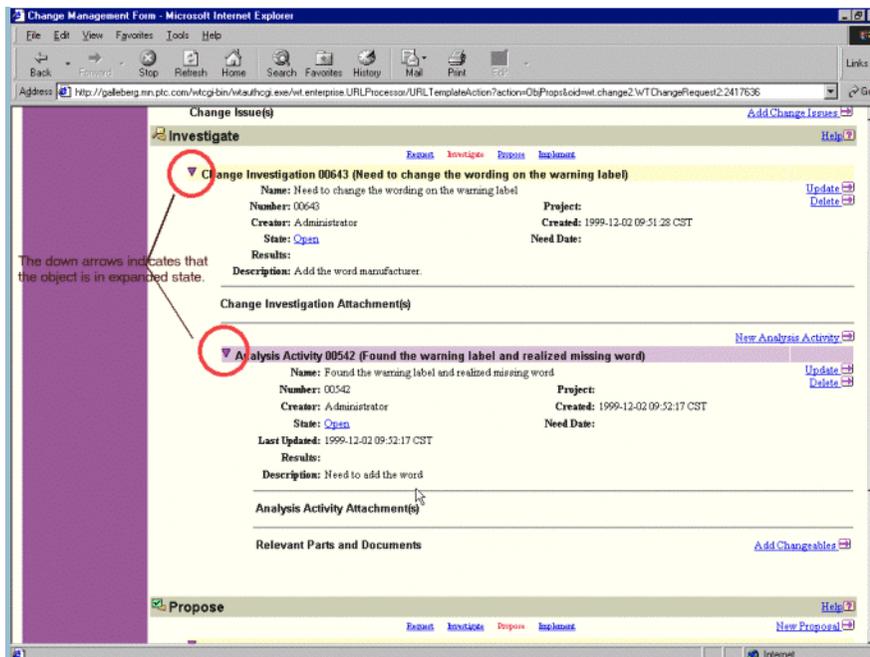
Generated by ChangeOrderProcessor/  
ChangeOrderSection.html.

Generated by ChangeOrderProcessor/  
UpdateChangeOrder.html.

## AnalysisActivityProcessor

Analysis activities are associated with change investigations and change proposals. Therefore, any change investigation or change proposal that is expanded, whether in view or update mode, triggers the display of the analysis activities table. From the templates `UpdateChangeInvestigation.html/UpdateChangeProposal.html` and `ViewChangeInvestigation.html/ViewChangeProposal.html`, a Windchill script call named `displayAnalysisActivityTable` invokes the method `displayAnalysisActivityTable` from `ChangeInvestigationProcessor` or `ChangeProposalProcessor`. A query is executed which finds all the analysis activities for the change investigation or change proposal. A header is printed for each row returned. If the object is to be displayed in expanded form, the context action is set according to the mode for that object, and a subtemplate service finds the appropriate processor, the `AnalysisActivityProcessor`. This processor, along with the template `ViewAnalysisActivity.html` or `UpdateAnalysisActivity.html`, depending on the mode, carries out the actions necessary to display the detail for the object.

The following figure shows the Investigate section in view mode with the change investigations and one analysis activity expanded.



The down arrows indicates that the object is in expanded state.

Generated by `ChangeInvestigationProcessor/ChangeInvestigationSection.html`

The processor/template combination `ChangeInvestigationProcessor/ViewChangeInvestigation.html` generate the attributes, attachments, and the list of Analysis Activities for this Change Investigation.

The processor/template combination `AnalysisActivityProcessor/ViewAnalysisActivity.html` are used to display the attributes, attachments, and changeables for an expanded Analysis Activity.

## ChangeActivityProcessor

Change activities are associated with change orders. Therefore, any change order that is expanded, whether in view or update mode, triggers the display of the change activities table. From ViewChangeOrder.html and UpdateChangeOrder.html, a Windchill script call named displayChangeActivityTable invokes the method displayChangeActivityTable from ChangeOrderProcessor. A query is executed which finds all the change activities for the change order. A header is printed for each row returned. If the object is to be displayed in expanded form, the context action is set according to the mode for that object, and a subtemplate service finds the appropriate processor, the ChangeActivityProcessor. This processor, along with the template ViewChangeActivity.html or UpdateChangeActivity.html, depending on the mode, carries out the actions necessary to display the detail for the object.

The following figure shows the Implement section in view mode with the change order and one change activity expanded.

The screenshot shows a web browser window titled "Change Management Form - Microsoft Internet Explorer". The address bar shows a URL from galeberg.mn.plc.com. The main content area displays a "Change Order 00742 (Changed the label)" with details such as Name, Number (00742), Creator (Administrator), Project, Created date (1999-12-02 09:53:49 CST), State (Open), and Description (Made the change to the label). Below this, there is a section for "Change Order Attachment(s)" and a "Change Activity 00841 (Removed the old label and added the new one)" which is expanded. The expanded activity shows its Name, Number (00841), Creator (Administrator), Project, Created date (1999-12-02 09:54:33 CST), Last Updated date (1999-12-03 09:25:38 CST), and Description (Removed the label). Underneath the activity, there is a table for "Change Activity Attachment(s)" with columns for File Name, Format, File Size, Last Updated, and Updated By. The table contains one entry: "ChangeManagementUI.doc" in Microsoft Word format, 206.5 KB, updated on 1999-12-03 09:25:43 CST by Administrator. Below the attachment table, there are sections for "Original Revisions (Parts and Documents)" and "New Revisions (Parts and Documents)", each with a table of columns: Number, Name, Version, Type, and Source. Both revision tables show a single entry with Number 9981779971, Name FS\_BRACKET, Version A, Type Part, and Source Make. The browser window also shows a sidebar on the left with a purple background and a "Links" button on the right.

The down arrows indicate that the objects are in expanded state.

The processor/template combination ChangeOrderProcessor/ViewChangeOrder.html generate the attributes, attachments, and the list of Change Activities for the Change Order.

The processor/template combination ChangeActivityProcessor/ViewChangeActivity.html generate the attributes, attachments, and changeables for the expanded Change Activity.

## Form Task Delegates

The links to perform a create, update, or delete action are handled by a Javascript function, processForm(), (shown below) which is included in the template DefaultChangeTask.html:

```
function processForm( context_url, context_action,
    query_name, query_value) {
if (verifyUserInput(context_action)){
    document.forms[0].action=context_url +?action=
        "+context_action+"&"+query_name+"="+query_value;
    document.forms[0].encoding="multipart/form-data";
    document.forms[0].submit();
}
}
```

The first parameter is the value HTML form action; that is, the URL where the information is to be submitted. The second parameter contains the value of the Windchill URL processor action. The third and fourth parameters contain the context object/context class and the object ID/class combination. For example, if a change order is in update mode, the Accept link will look similar to the following:

```
<A HREF="javascript:processForm('http://galleberg.mn.ptc.com/
    wtcgi-bin/wtauthcgi.exe/wt.enterprise.URLProcessor/processForm',
    'UpdateChangeObject', 'oid',
    'OR:wt.change2.WTChangeOrder2:2521002' )" >
Accept 
<IMG SRC="http://galleberg.mn.ptc.com/Windchill/wt/clients/
    images/actnlink.gif" ALT="Accept changes" BORDER=0></A>
```

The remainder of this section describes the task delegate for each of the three actions:

- CreateChange<object>Delegate
- UpdateChange<object>Delegate
- DeleteChange<object>Delegate

## CreateChange<object>Delegate

In create mode, the Accept link triggers the appropriate form task delegate for the object to be created. The Accept link contains the action CreateChangeObject and the class of the type of object to create. Following is an example of the structure of an Accept link for creating a new change order:

```
<A HREF=" javascript:processForm('http://SiteWindchillServer.Com/wtcgi-bin/wtauthcgi.exe/wt.enterprise.URLProcessor/processForm', 'CreateChangeObject', 'class', 'wt.change2.WTChangeOrder2' )">Accept 
<IMG SRC="http://SiteWindchillServer.Com/windchill/wt/clients/images/actnlink.gif" ALT="Accept changes" BORDER=0></A>
```

The delegate gets the values for the input data, creates a new instance of the class, and then saves the new object. Upon completing these tasks, the change management user interface is repainted with the new object expanded and scrolled into view.

## UpdateChange<object>Delegate

In update mode, the Accept link triggers the appropriate form task delegate for the object to be updated. The Accept link contains the action UpdateChangeObject and the oid of the object to update. Following is an example of the structure of an Accept link for updating a change order:

```
<A HREF=" javascript:processForm('http://SiteWindchillServer.Com/wtcgi-bin/wtauthcgi.exe/wt.enterprise.URLProcessor/processForm', 'UpdateChangeObject', 'oid', 'OR:wt.change2.WTChangeOrder2:2417705' )">Accept 
<IMG SRC="http://SiteWindchillServer.Com/windchill/wt/clients/images/actnlink.gif" ALT="Accept changes" BORDER=0></A>
```

The delegate gets the values for the input data and then saves the object. Upon completing these tasks, the change management user interface is repainted reflecting the changes to the object.

## DeleteChange<object>Delegate

In view mode, the Delete link triggers the appropriate form task delegate for the object to be deleted. The Delete link contains the action DeleteChangeObject and the OID of the object to delete. Following is an example of the structure of a Delete link for deleting a change order:

```
<A HREF=" javascript:processForm('http://SiteWindchillServer.Com/wtcgi-bin/wtauthcgi.exe/wt.enterprise.URLProcessor/processForm', 'DeleteChangeObject', 'oid', 'OR:wt.change2.WTChangeOrder2:2417705' )">Delete 
<IMG SRC="http://SiteWindchillServer.Com/windchill/wt/clients/images/actnlink.gif" ALT="Delete the Change Order" BORDER=0></A>
```

Upon completing the tasks, the change management user interface is repainted without the deleted object.

## Changing the Look of the User Interface

Changing the look of the user interface can be as simple as changing an HTML template; or it may involve subclassing template processors and changing properties files, HTML templates, and resource bundles.

The properties file used for the change management system is located under the codebase directory in `wt/change2/change2.properties`. If instructed to make a change to a properties file, follow the pattern of the steps in chapter 5, *Customizing service.properties*.

The HTML templates are located under the source code directory in `wt/clients/change2`. If instructed to make a change to an HTML template, follow the instructions given in chapter 7, *Customizing the HTML Client*. To find the template used to generate a particular element, see the processor/template reference tables later in this chapter.

The resource bundles for change management exist in the following locations:

- `wt/change2/Change2Resource_*.java`
- `wt/clients/Change2/Change2RB_*.java`
- `wt/change2/htmlclient/htmlclientResource_*.java`

If instructed to make changes to resource bundles, follow the pattern of the steps in the section named *Changing Display Names for Classes, Attributes, and Association Roles* in chapter 4, *Customizing GUIs*.

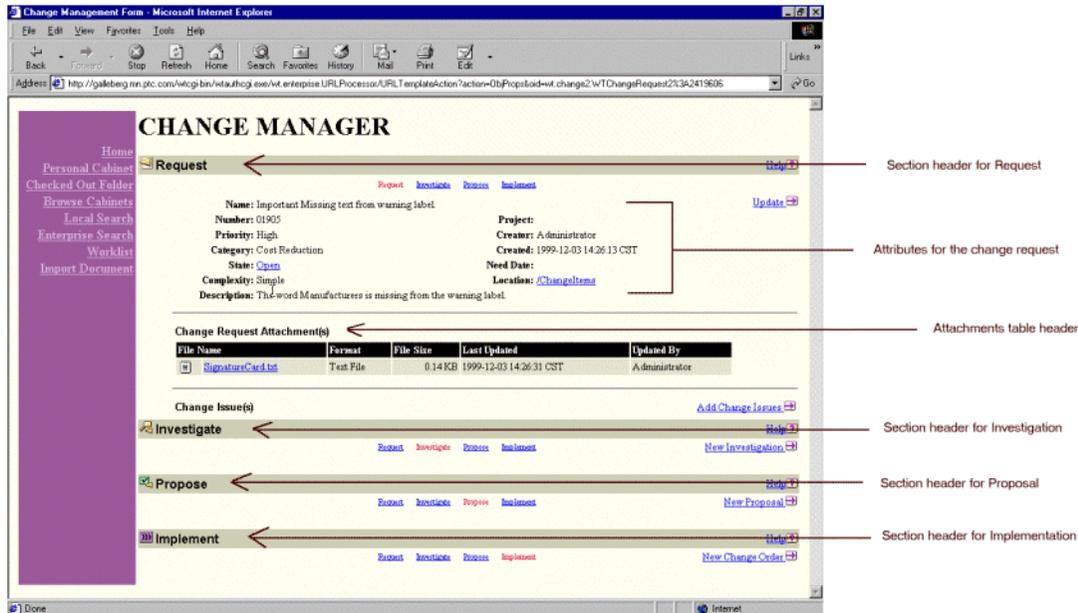
The following sections describe how to change the appearance of elements on the change management page -- that is, text, font, and background colors -- and how to remove or change actual elements of the page -- that is, whole sections, action links, expand and collapse functionality, and content and changeables tables.

### Changing Text

This section describes how to change the text that is displayed for section headers, table headers, and attribute labels. The text displayed in the user interface is generated by script calls in the templates. Resource bundles are used to produce localized text.

## Changing the Text for Section Headers

Section headers are identified in the user interface by a colored background that contains the name of the task as well as the icon associated with the object that relates to that task. The four sections are Request, Investigate, Propose, and Implement.



To change the text for a section header, you must change the resource bundle that contains the text used for headers, `wt\clients\change2\change2RB.java`. In this resource bundle, you will see entries for each of the resource keys, as shown below in the left column. Make the changes to the desired text entries in the right column.

```
{REQUEST,          "Request" },
{INVESTIGATE,     "Investigate"},
{PROPOSE,         "Propose" },
{IMPLEMENT,       "Implement" },
```

A Windchill script call is used to insert each header into the generated page. Following is the script call found in `ChangeInvestigationSection.html` that generates the Investigate header text:

```
<SCRIPT LANGUAGE=Windchill>
<!--
  getLocalizedMessage resourceKey=INVESTIGATE
    resourceClass=wt.clients.change2.Change2RB
-->
</SCRIPT>
```

The script calls for the other sections are similar. The `resourceKey` parameter value indicates the entry in the resource bundle which contains the text to display.

## Changing the Text for Table Headers

Each change object has several types of attachments and associations. For example, a change request can have attachments and change issues. These elements are listed in tables in the Request section. The preceding figure (in the section titled Changing the Text for Section Headers) shows the header for a change request attachments table. To change the headings for these tables, you must change the appropriate resource bundle.

To find the appropriate resource class and resource key, find the Windchill script call in the HTML template where the table header is generated. For example, the attachments table heading for Change Request Attachment(s) is generated in `wt/clients/change2/ViewChangeRequest.html` and `wt/clients/change2/UpdateChangeRequest.html`. You can look in either of these templates to find the script call associated with generating the heading. The script call will look similar to the following:

```
<SCRIPT LANGUAGE=Windchill>
<!--
  getLocalizedMessage resourceKey=CHANGE_REQUEST_ATTACHMENTS
    resourceClass=wt.clients.change2.Change2RB
-->
</SCRIPT>
```

By changing the value of the resource key `CHANGE_REQUEST_ATTACHMENTS` in resource class `wt.clients.change2.Change2RB`, you can change the table header that is displayed for the change request attachments table in both view and update mode.

## Changing the Text for Object Attribute Labels

The attributes for an object are displayed using an HTML table. The values of the labels for most of these attributes are located in the resource bundle `wt/change2/change2Resource.java`. Each object has an entry for each attribute. If two objects have the same attribute name, each must have its own entry for the display of the attribute name; for example:

```
{ "WTChangeRequest2.name.DisplayName", "Name" }
.
.
{ "WTChangeProposal.name.DisplayName", "Name" }
```

The templates contain Windchill script calls for each attribute that is displayed. Most of these attributes use the following syntax:

```
<SCRIPT LANGUAGE=Windchill>
<!--
  objectPropertyName propertyName=name
-->
</SCRIPT>
```

For a change proposal, this script call (found in ViewChangeProposal.html and UpdateChangeProposal.html) references the following entry from the resource bundle wt\change2\change2Resource.java:

```
{ "WTChangeProposal.name.DisplayName", "Name" }
```

The value Name is displayed as the label for the name field.

## Changing Font

This section describes how to change the font attributes for several elements. These attributes include size, face, and color. These attributes are applied to HTML elements within the templates so most changes can be made quickly and easily. Information displayed within a table requires changes to table service script calls within the templates.

### Changing the Font for Section Headers

The font for section headers is set in the HTML templates. To change this font, you must find the appropriate template and change the HTML <FONT> tag. Following is the <FONT> tag used to indicate the face and size of the Investigate section header text. This block of code can be found in the ChangeInvestigationSection.html template.

```
<FONT face="arial, helvetica" size="3">
  <B>
  <SCRIPT LANGUAGE=Windchill>
  <!--
    getLocalizedMessage resourceKey=INVESTIGATE
      resourceClass=wt.clients.change2.Change2RB
  -->
  </SCRIPT>
  </B>
</FONT>
```

### Changing the Font for Table Headers

The font for table headers is also indicated in HTML templates. To change this font, you must find the appropriate template and change the <FONT> tag used for the table header you want to change.

```
<TD>
  <FONT size=2 face="arial, helvetica">
  <B>
  <SCRIPT LANGUAGE=Windchill>
  <!--
    getLocalizedMessage resourceKey=ANALYSIS_ACTIVITY_ATTACHMENTS
      resourceClass=wt.clients.change2.Change2RB
  -->
  </SCRIPT>
  </B>
  </FONT>
</TD>
```

## Changing the Font for Object Attribute Labels

Each attribute for an object has a corresponding font tag in the HTML templates. To change this font, you must find the template where you want to change the font for the attribute. Following is an example of a block of HTML code from a template that is used to display the property name for creator. The font face used is the default that is set for the browser. The size is set to 2.

```
<TD width="35%" valign=top>
<FONT size=2>
<SCRIPT LANGUAGE=Windchill>
  <!--
  objectPropertyName propertyName=creator
  -->
</SCRIPT>
</FONT>
</TD>
```

## Changing Background Color

This section describes how to change the color of backgrounds on the page, the section headers, and the change object headers.

### Changing the Background Color for the Page

The background color for the page and the navigation bar is specified in the wt\clients\change2\ChangeManagementForm.html template. To change this color, locate the <BODY> tag in this template and change the BGCOLOR attribute value to the desired color.

```
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0000EE"
  VLINK="#551A8B" ALINK="#FF0000" FONT FACE=Arial
  LANGUAGE=javascript onload="return window_onload()"
  onUnload="closeFolderWindow()">
```

To change the background color of the navigation bar (on the left side of the page), find the section similar to the following and change the value of the BGCOLOR attribute in the <TD> tag:

```
<TD ALIGN=RIGHT VALIGN=TOP WIDTH=15% BGCOLOR=#9B599B
  CELLPADDING=0 CELLSPACING=0>
  <P>&nbsp;</P>
  <TABLE WIDTH=100% CELLPADDING=0 CELLSPACING=0>
    <TR ALIGN=RIGHT>
      <TD>
        <SCRIPT LANGUAGE=Windchill>
          <!--
          createGlobalNavigationBar currentPage=CMForm
          -->
        </SCRIPT>
      </TD>
    </TR>
  </TABLE>
</TD>
```

## Changing the Background Color for Section Headers

Each major section, except the Request section, has its own template. The Request section is unique in that it is defined by three templates. For each of the objects, open the appropriate template (or templates, in the case of the Request section) and search for the section that is similar to the following code:

```
<TR bgcolor=#ccccb4>
  <TD height="5%">
    <A name="Investigate">
      <SCRIPT language=windchill>
        getClassIcon object_class=wt.change2.WTChangeInvestigation
      </SCRIPT>
      <FONT face="arial, helvetica" size="3">
        <B>
          <SCRIPT LANGUAGE=Windchill>
            <!--
getLocalizedMessage resourceKey=INVESTIGATE
resourceClass=wt.clients.change2.Change2RB
-->
          </SCRIPT>
        </B>
      </FONT>
    </A>
  </TD>
  <TD width="10%" align=right>
    <FONT size=2>
      <SCRIPT LANGUAGE=Windchill>
        <!--
addHTMLHelpLink HelpContext=VIEW_INVESTIGATION
HelpLabel=VIEW_INVESTIGATION
HelpLabelResource=wt.clients.change2.Change2RB
-->
      </SCRIPT>
    </FONT>
  </TD>
</TR>
```

The background color is specified by the row tag, `<TR bgcolor=#ccccb4>`. You make the change for the background color in this tag. Any hexadecimal or text value is allowed for the value (red, blue, #dddfef).

## Changing the Background Color for Change Object Headers

The background color for the change object headers is specified in the table service calls that display the list of objects. The color is specified for two columns. The following code is an example of the code used to display a list of analysis activities:

```
<SCRIPT LANGUAGE=Windchill>
  <!--
    displayAnalysisActivityTable
    tableService action=SETSERVICENAME SERVICENAME=
      wt.change2.htmlclient.ImageLinkStringCellComponent
    tableService action=ADDCOLUMN NAME=displayIdentity
      COLUMNCLASS=wt.change2.htmlclient.ExpandableTableColumn
    tableService action=setColumnAttributes COLUMNNUMBER=0
      BOLD=true td.bgcolor=#d8bfd8 FONTSIZE=2
      FONTFACE=arial,Helvetica
    tableService action=ADDCOLUMN NAME=updateColumn
      COLUMNCLASS=wt.templateutil.table.HTMLTableColumn
    tableService action=setColumnAttributes COLUMNNUMBER=1
      td.BGCOLOR=#d8bfd8 td.align=right
      font.size=2 font.face=arial,Helvetica
    tableService action=setTableAttributes table.ALIGN=right
      table.WIDTH=95% table.cellspacing=1
      table.cellpadding=0
    tableService action=SHOW
  -->
```

To change the background color, find the similar section and change the value of the BGCOLOR parameter to the new color. Either hexadecimal or text values can be used.

## Removing or Changing Parts of the User Interface

### Removing Change Investigation, Change Proposal, or Change Order (Implementation) Sections

If you will not be using the change investigation, change proposal, or change order objects, you can remove them by changing the DefaultChangeTask.html template. This template contains four script calls, each one triggering the generation of a particular section. Removing a script call prevents the corresponding section from being generated. For example, if you do not want the change investigation section generated, you must remove the following Windchill script call:

```
<SCRIPT LANGUAGE=Windchill>
  <!--
    displayChangeInvestigationSection
  -->
</SCRIPT>
```

The other sections of the page continue to be generated.

## Removing Analysis Activity or Change Activity Sections

A Windchill script call is used to display the list of analysis activities/change activities. The script includes calls to the table service. To prevent these objects from being displayed, remove the entire script call, which is similar to the following:

```
<TR>
  <TD colspan=2>
    <SCRIPT LANGUAGE=Windchill>
      <!--
        displayAnalysisActivityTable
        tableService action=SETSERVICENAME
          SERVICENAME=wt.change2.htmlclient.
            ImageLinkStringCellComponent
        tableService action=ADDCOLUMN NAME=displayIdentity
          COLUMNCLASS=wt.change2.htmlclient.ExpandableTableColumn
        tableService action=setColumnAttributes COLUMNNUMBER=0
          BOLD=true td.bgcolor=#d8bfd8 FONTSIZE=2
          FONTFACE=arial,helvetica
        tableService action=ADDCOLUMN NAME=updateColumn
          COLUMNCLASS=wt.templateutil.table.HTMLTableColumn
        tableService action=setColumnAttributes COLUMNNUMBER=1
          td.BGCOLOR=#d8bfd8 td.align=right font.size=2
          font.face=arial,helvetica
        tableService action=setTableAttributes table.ALIGN=right
          table.WIDTH=95% table.cellspacing=1 table.cellpadding=0
        tableService action=SHOW
      -->
    </SCRIPT>
  </TD>
</TR>
```

## Removing or Changing an Action Link

Each link that is displayed is generated using a Windchill script call. To prevent a particular link from being generated, you must remove the script call from the appropriate template.

If you want to change the functionality of an action link, or change the object type that the action is performed on, follow the pattern of the steps in the following example.

This example shows how to change the action links for creating a new object. Specifically, the user interface action link New Change Investigation must be changed to allow the creation of the new type of object.

This example assumes the following:

- You have subclassed `wt.change2.WTChangeInvestigation` and named it `site.change2.SiteChangeInvestigation`.
- You have copied the templates `ChangeInvestigationSection.html` and `ViewChangeInvestigation.html`, and named them

SiteChangeInvestigationSection.html and SiteViewChangeInvestigation.html respectively.

1. The process and template reference tables later in this chapter show that the New Change Investigation link is generated from ChangeInvestigationSection.html. The script call indicates the class type that is to be created.

```
<SCRIPT LANGUAGE=Windchillgt;  
  <!--  
    getNewChangeActivityLink  
      class=wt.change2.WTChangeInvestigation  
      resourceKey=NEW_CHANGE_INVESTIGATION  
      resourceClass=wt.change2.htmlclient.htmlclientResource  
      altKey=NEW_CHANGE_INVESTIGATION_ALT  
      img=wt/clients/images/actnlink.gif  
  -->  
</SCRIPT>
```

Change the value for the class parameter from wt.change2.WTChangeInvestigation to the new class, site.change2.SiteChangeInvestigation.

2. A second script call in this template must also be modified. This script call, shown below, determines if the create\_class hidden form field contains the matching class for this section.

```
<SCRIPT LANGUAGE=Windchillgt;  
  <!--  
    displayCreateNewObject class=wt.change2.WTChangeInvestigation  
  -->  
</SCRIPT>
```

Again, change the value for the class parameter to the new class, site.change2.SiteChangeInvestigation.

3. When in create mode in the user interface, the user is presented with two options: Accept or Cancel. The URL associated with the Accept link includes the type of object that is to be created. In the following script call, found in the template UpdateChangeInvestigation.html (noted in the processor/template reference tables), change the class parameter value from wt.change2.WTChangeInvestigation to the new class, site.change2.SiteChangeInvestigation.

```
<SCRIPT LANGUAGE=Windchillgt;  
  <!--  
    getAcceptLink class=wt.change2.WTChangeInvestigation  
      resourceClass=wt.change2.htmlclient.htmlclientResource  
      altKey=ACCEPT_CHANGES img=wt/clients/images/actnlink.gif  
  -->  
</SCRIPT>
```

## Removing the Expand and Collapse Functionality

To remove the expand and collapse functionality, follow the pattern of the steps in the following example. In this example, assume that change orders and change activities should always be expanded, and remove the ability to collapse these objects. Also assume that the change investigation, analysis activities, and change proposals are not being used.

1. Subclass `wt.change2.htmlclient.ExpandableTable` and name it `SiteExpandableTable`.
2. Override the method `printRow`.
3. Add the following code to the `printRow` method. Note that `super.printRow` is also being called.

```
String object_reference = null;
WTOBJECT row_source = null;

// Get the object that represents the row from the table model.
RowDataTableModel tm = (RowDataTableModel)getTableModel();
row_source =
    (WTOBJECT)tm.getRowDataObjects().elementAt(rowNumber);

// Get the oid from the reference factory.
try {
    object_reference = new
        ReferenceFactory().getReferenceString((Persistable)row_source);
} catch (WTEException wte) {
//handle the exception
}

//Add the form data (HTML hidden form field) that indicates
//that the object display should be expanded
formData.put(object_reference, "expand");

//Call the parent printRow method, to display the information
return super.printRow(rowNumber, value, formData );
```

As explained in the Expand and Collapse Functionality section earlier in this chapter, hidden form fields are used to indicate which objects should be in expanded state. The preceding code adds this form field to the form data so that each time the page is repainted, the change orders and change activities are expanded.

4. Subclass `wt.change2.htmlclient.ChangeOrderProcessor`.
5. Override the method `initiateExpandableTable`.
6. Add the following code to `initiateExpandableTable`:

```
//if there are no query results simply write out a blank space
//and return.
if (!qr.hasMoreElements()){
    PrintWriter out = getPrintWriter(os, locale);
    out.print("&nbsp;");
```

```

        out.flush();
        return;
    }

    Vector resultVector = new Vector();
    while (qr.hasMoreElements()) {
        resultVector.addElement(qr.nextElement());
    }

    // Create the HTMLTable subclass, SiteExpandableTable.
    // The table columns can be
    // set in the html template.
    SiteExpandableTable table = new
        SiteMyExpandableTable(resultVector, this, locale, os );
    table.init(null, null, null, null, null);
    table.setOutputStream(os);
    table.setLocale(locale);
    getHTMLTableService().setHtmlTable(table);

```

This is the same code that is used in the `initiateExpandableTable` method in `ChangeOrderProcessor`, with the exception of the instance of the `ExpandableTable`. Instead, an instance of `SiteExpandableTable` is created.

7. Add the reference to the new processor to your site's properties file as follows:

```

wt.services/svc/default/wt.enterprise.TemplateProcessor/
  ChangeOrderSection/java.lang.Object/0=
  wt.change2.htmlclient.SiteChangeOrderProcessor/duplicate
wt.services/svc/default/wt.enterprise.TemplateProcessor/
  ViewChangeObject/wt.change2.ChangeOrderIfc/0=
  wt.change2.htmlclient.SiteChangeOrderProcessor/duplicate
wt.services/svc/default/wt.enterprise.TemplateProcessor/
  UpdateChangeObject/wt.change2.ChangeOrderIfc/0=
  wt.change2.htmlclient.SiteChangeOrderProcessor/duplicate
wt.services/svc/default/wt.enterprise.TemplateProcessor/
  CreateChangeObject/wt.change2.ChangeOrderIfc/0=
  wt.change2.htmlclient.SiteChangeOrderProcessor/duplicate

```

8. Change the parameter values in the table service calls in the HTML templates to use the standard HTML components or your site-specific classes, if you have customized them. In `ChangeOrderSection.html`, `ViewChangeOrder.html`, and `UpdateChangeOrder.html`, make the following changes:

- a. Change the parameter:

```

tableService action=SETSERVICENAME
  SERVICENAME=wt.change2.htmlclient.ImageLinkStringCellComponent

```

to the following:

```

tableService action=SETSERVICENAME
  SERVICENAME=wt.templateutil.components.HTMLTableCellComponent

```

b. Change the parameter:

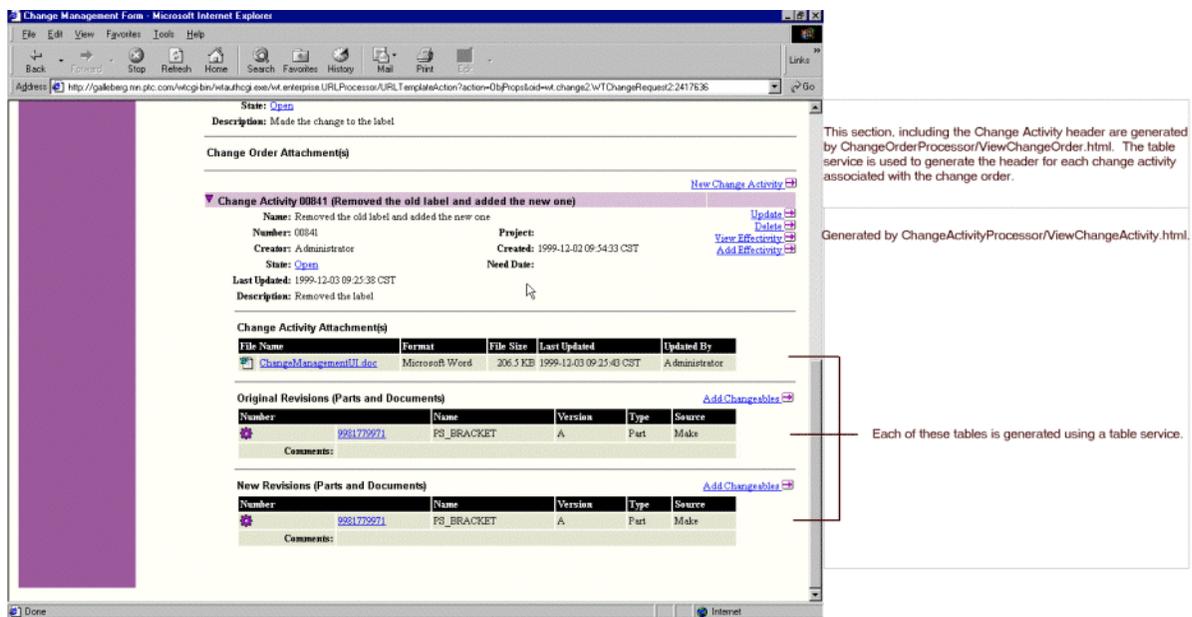
```
tableService action=ADDCOLUMN NAME=displayIdentity
COLUMNCLASS=wt.change2.htmlclient.ExpandableTableColumn
```

to the following:

```
tableService action=ADDCOLUMN NAME=displayIdentity
COLUMNCLASS=wt.templateutil.table.HTMLTableColumn
```

## Content and Changeables

The content and changeables tables are generated using calls to table service. Changing the attributes that are displayed in the table may be a simple change in the HTML template that generates the table. The following figure illustrates a change activity and the processor/template used to generate the page:



The following example explains the calls used to generate the Change Activity Attachment(s) table shown in the preceding figure. These lines of code were taken from ViewChangeActivity.html.

The first script call is used to get URL data associated with the object. Although this is supported, it does not apply to the current version of Change Management.

```
<TR valign=top>
  <TD>
    <SCRIPT LANGUAGE=Windchill>
    <!--
      tableService action=initializeContents
      tableService action=initURLDataTable
        URLDATAATTRIBUTES=urlLocation,description
      tableService action=setHeaderAttributes
```

```

        name=ALL font.SIZE=2 th.align=center
        font.color=white
    tableService action=setColumnAttributes
        name=ALL font.SIZE=2 td.bgcolor=#e3e3cf
        td.align=left
    tableService action=setTableAttributes
        table.width=90% table.cellspacing=2
        table.cellpadding=2
    tableService action=show
-->
</SCRIPT>

```

The following script call is used to display the attachments table for the object. (The line numbers in bold on the left correspond to notes following the code sample; they are not part of the code.)

```

<SCRIPT LANGUAGE=Windchill>
<!--
Line 1 tableService action=INITAPPLICATIONDATATABLE
        //APPLICATIONDATAATTRIBUTES=fileName,format,fileSize,
        //modifyTimestamp,createdBy
Line 2 tableService action=setHeaderFromResource POSITION=4
        RESOURCEBUNDLE=wt.enterprise.enterpriseResource
        RESOURCEKEY=UPDATED_BY
Line 3 tableService action=setHeaderAttributes name=ALL
        font.SIZE=2 th.align=left font.color=white
Line 4 tableService action=setColumnAttributes name=ALL
        font.SIZE=2 td.bgcolor=#e3e3cf td.align=left
Line 5 tableService action=setTableAttributes table.width=90%
Line 6 tableService action=show
-->
</SCRIPT>
</TD>
</TR>

```

**Line 1** indicates the type of data to be displayed and the attributes to be displayed.

**Line 2** sets the column header text according to values in the indicated resource bundle.

**Line 3** sets the attributes values.

**Line 4** sets the column attributes.

**Line 5** sets the table attributes.

**Line 6** generates the table given the preceding information.

## Processor/Template Reference Tables

Each time the change management form is generated, two HTML templates are used: ChangeManagementForm.html and DefaultChangeTask.html. Each change object has an associated template for each mode (view, update, and create). The Investigate, Propose, and Implement sections each have a template associated with them. There is also a template used to include the attachments table in update mode.

Use the following tables to find the template/processor that includes the element you want to modify.

### Action Links

Object	New	Template
Change Investigation	New Change Investigation	ChangeInvestigationSection.html/ ChangeInvestigationProcessor
Change Proposal	New Change Proposal	ChangeProposalSection.html/ ChangeProposalProcessor
Change Order	New Change Order	ChangeOrderSection.html/ ChangeOrderProcessor
Change Activity	New Change Activity	ViewChangeOrder.html/ ChangeOrderProcessor
Analysis Activity	New Analysis Activity	ViewChangeInvestigation.html/ ChangeInvestigationProcessor
		ViewChangeProposal.html/ ChangeProposalProcessor
Object	Update	Template/Processor
Change Request		ViewChangeRequest.html/ ChangeRequestProcessor
Change Investigation		ViewChangeInvestigation.html/ ChangeInvestigationProcessor
Change Proposal		ViewChangeProposal.html/ ChangeProposalProcessor
Change Order		ViewChangeOrder.html/ ChangeOrderProcessor

Change Activity		ViewChangeActivity.html/ ChangeActivityProcessor
Analysis Activity		ViewAnalysisActivity.html/ AnalysisActivityProcessor
Object	Accept/Cancel	Template
Change Request		UpdateChangeRequest.html
Change Investigation		UpdateChangeInvestigation.html/ ChangeInvestigationProcessor
Change Proposal		UpdateChangeProposal.html/ ChangeProposalProcessor
Change Order		UpdateChangeOrder.html/ ChangeOrderProcessor
Change Activity		UpdateChangeActivity.html/ ChangeActivityProcessor
Analysis Activity		UpdateAnalysisActivity.html/ AnalysisActivityProcessor
Object	Delete	Template
Change Investigation		ViewChangeInvestigation.html/ ChangeInvestigationProcessor
Change Proposal		ViewChangeProposal.html/ ChangeProposalProcessor
Change Order		ViewChangeOrder.html/ ChangeOrderProcessor
Change Activity		ViewChangeActivity.html
Analysis Activity		ViewAnalysisActivity.html/ AnalysisActivityProcessor
Object	View/Add Effectivity	Template/Processor
Change Order		ViewChangeOrder.html/ ChangeOrderProcessor
Change Activity		ViewChangeActivity.html

## Section Headers

Object	Header	Template/Process
Change Request	Request	ViewChangeRequest.html/ ChangeRequestProcessor
		UpdateChangeRequest.html/ ChangeRequestProcessor
		CreateChangeRequest.html/ ChangeRequestProcessor
Change Investigation	Investigate	ChangeInvestigationSection.html/ ChangeInvestigationProcessor
Change Proposal	Propose	ChangeProposalSection.html/ ChangeProposalProcessor
Change Order	Order	ChangeOrderSection.html/ ChangeOrderProcessor

## Expandable/Collapseable Objects

Object	Table Server	Template/Processor
Change Investigation		ChangeInvestigationSection.html/ ChangeInvestigationProcessor
Change Proposal		ChangeProposalSection.html/ ChangeProposalProcessor
Change Order		ChangeOrderSection.html/ ChangeOrderProcessor
Change Activity		UpdateChangeOrder.html/ ChangeOrderProcessor
		ViewChangeOrder.html/ ChangeOrderProcessor
Analysis Activity		UpdateChangeInvestigation.html/ ChangeInvestigationProcessor
		ViewChangeInvestigation.html/ ChangeInvestigationProcessor

Object	Table Server	Template/Processor
		UpdateChangeProposal.html/ ChangeProposalProcessor
		ViewChangeProposal.html/ ChangeProposalProcessor

## Adding an Attribute to WTChangeRequest2

This customization scenario adds a new String attribute named "justification" to the change request object. The new attribute could be used to record a justification (financial or strategic) of why a change request should be opened. Adding an attribute to any other change object would be similar.

### Directory Structure

Save the Java files you create in the /Windchill/src/customization/SiteChange2 directory or its subdirectories and then Visual Café will compile the class files into the /Windchill/codebase/customization/SiteChange2 directory or its subdirectories.

You can save the HTML files and properties files in the /Windchill/src/customization/SiteChange2 directory or its subdirectories and then manually copy them to the /Windchill/codebase/customization/SiteChange2 directory or its subdirectories. An alternative would be to save them in the /Windchill/codebase/customization/SiteChange2 directory or its subdirectories; then you do not have to copy them manually.

You can make copies of existing HTML files to use as a starting point by copying from the /Windchill/codebase/templates/change2 directory.

The following directory structure is suggested. You will need to create many of these directories. If you choose a different directory structure, some steps in this customization example will need to be modified.

/Windchill/codebase/customization/SiteChange2/

SiteChange2.properties  
SiteChange2Resource.class  
SiteChangeRequest.class  
SiteChangeRequest.ClassInfo.ser

/Windchill/codebase/customization/SiteChange2/clients/

SiteChangeRequestTaskDelegate.class  
SiteCreateChangeRequest.html  
SiteUpdateChangeRequest.html  
SiteViewChangeRequest.html  
SiteChangeManagementHome.html

/Windchill/codebase/customization/SiteChange2/htmlclient/

SiteCreateChangeRequestDelegate.class  
SiteUpdateChangeRequestDelegate.class

/Windchill/src/customization/SiteChange2/

SiteChange2.mData  
SiteChange2.properties  
SiteChange2Resource.java  
SiteChangeRequest.java

/Windchill/src/customization/SiteChange2/clients/

clients.cat  
clients.mData  
SiteChangeRequestTaskDelegate.java  
SiteCreateChangeRequest.html  
SiteUpdateChangeRequest.html  
SiteViewChangeRequest.html

/Windchill/src/customization/SiteChange2/htmlclient/

htmlclient.cat  
htmlclient.mData  
SiteCreateChangeRequestDelegate.java  
SiteUpdateChangeRequestDelegate.java

/Windchill/src/customization/SiteChange2/Rose98/

SiteChange2.cat  
SiteChange2.mdl

/Windchill/src/customization/SiteChange2/Vcafe/

SiteChange2.cdb  
SiteChange2.ve2  
SiteChange2.vep  
SiteChange2.vpj

## Work Overview

This section gives an overview of the tasks that you will perform in the sections that follow. Do not begin these tasks yet; more detailed instructions are given in the following sections.

1. Save a copy of the Rose model `wt.WTDesigner.mdl` named `SiteChange2.mdl` so that you can keep your modeling changes separate from the main Windchill model.
2. Create a `SiteChange2.properties` file and make a reference to it from `wt.properties` so that you can keep your properties entries separate from the main Windchill properties.
3. Make a subclass of `WTChangeRequest2` (name the subclass `SiteChangeRequest`) and add an attribute of type `String` named "justification". Make it public so that code generation will generate the public accessor methods `getJustification` and `setJustification`.
4. Make a new Change Management home page that will create a `SiteChangeRequest` instead of a `WTChangeRequest2`.
5. Copy `CreateChangeRequest.html`, `UpdateChangeRequest.html`, and `ViewChangeRequest.html`. Name the copies `SiteCreateChangeRequest.html`, `SiteUpdateChangeRequest.html`, and `SiteViewChangeRequest.html`, and add new HTML as needed. These will be the HTML templates used to display your `SiteChangeRequest` objects when they are created, updated, or viewed.
6. Subclass delegates `CreateChangeRequestDelegate` and `UpdateChangeRequestDelegate`. Name them `SiteCreateChangeRequestDelegate` and `SiteUpdateChangeRequestDelegate`. You must edit the `SiteChange2.properties` file to cause the update and create delegates you just created to be used with a `SiteChangeRequest` object.
7. Subclass the delegate `TaskDelegate`. Name it `SiteChangeRequestTaskDelegate`. This delegate is automatically invoked, so you do not need to edit the `SiteChange2.properties` file to invoke it.
8. Create `SiteChange2Resource.java` for error and customization messages, and so on. Copy an existing resource file and remove (or comment out) unneeded sections; this ensures you get the correct format.

The following sections, separated by function, describe how to perform these steps in more detail.

## Rational Rose Tasks

1. Save a copy of the wt.WTDesigner.mdl named SiteChange2.mdl. Note that if wt.WTDesigner.mdl is read-only, you may find that your SiteChange2.mdl is initially read-only. You can fix this by choosing Open from the File menu and reselecting SiteChange2.mdl.
2. Create the customization package within the Logical View folder; create the SiteChange2 package within the customization package; create the clients package and the htmlclient package within the SiteChange2 package.
3. Within the SiteChange2 package, subclass SiteChangeRequest from WTChangeRequest2. Add the new attribute "justification" and add methods "checkAttributes" and two "SiteChangeRequest" constructors. Be sure to make the justification attribute a String with no default value. It is best to copy the two methods from WTChangeRequest2 and then change the names of the constructors and of the return types (from WTChangeRequest2 to SiteChangeRequest) so that you get all the correct settings on the Windchill tab.
4. Within the clients package, subclass SiteChangeRequestTaskDelegate from TaskDelegate and copy the "launchCreateTask" method from TaskDelegate. (In a later step, you will need to change the copied method in Visual Café to make it work for a SiteChangeRequest object.)
5. Within the htmlclient package, perform the following actions:
  - a. Subclass SiteCreateChangeRequestDelegate from CreateChangeRequestDelegate and override the methods "setAttributes" and "createChangeItem" by copying them from CreateChangeRequestDelegate.
  - b. Subclass SiteUpdateChangeRequestDelegate from UpdateChangeRequestDelegate and override the "updateAttributes" method by copying it from UpdateChangeRequestDelegate.

**Note:** In all cases, you will need to change the copied methods so they are no longer abstract; this means changing the stereotype and a property on the Windchill tab. Whenever a method you copy is abstract and you want the new method to be non-abstract, you must change both the stereotype (make it have no stereotype) and change the Abstract property to FALSE on the Windchill tab.
6. Generate the files from your model by selecting **Tools > Windchill > System Generation** from the Rational Rose menu.

## SQL Tasks

The only SQL task is to run the following script that was generated by Rational Rose:

```
\Windchill\db\sqlcustomization\SiteChange2\Make_SiteChange2.sql
```

This script creates the tables needed to support your new objects. Note that if you rerun this script, you may get errors creating primary keys, but they can be ignored.

## Visual Café Tasks

You can use any Java development environment you want. This example assumes Visual Café is used, but others development environments work just as well.

1. Include all the files generated from Rose in your Visual Café project.
2. For SiteChangeRequest:
  - The checkAttributes method should call super.checkAttributes and it should check the justification attribute. If the justification is null or an empty string, issue an error message that is created below in SiteChange2Resource.java (called NO\_CR\_JUSTIFICATION):

```
super.checkAttributes();

    if ((getJustification() == null)
        || (getJustification().equals(""))) {
throw new InvalidAttributeException(RESOURCE,
    SiteChange2Resource.NO_CR_JUSTIFICATION, null );
    }
```

- The two constructors should be generated for you by Rational Rose as follows:

For newSiteChangeRequest ():

```
SiteChangeRequest instance = new SiteChangeRequest ();
instance.initialize();
return instance;
```

For newSiteChangeRequest (name):

```
SiteChangeRequest instance = new SiteChangeRequest ();
instance.initialize( name );
return instance;
```

- The initialize method (only a stub was generated for you) should call super.initialize (name):

```
super.initialize( name );
```

If you forget to do this, you will get a message at runtime that a name must be provided for the change request.

3. For `SiteChangeRequestTaskDelegate`, code the `launchCreateTask` method as follows:

```
try {
    super.launchCreateTask();
    Properties urlParameter = new Properties ();
    urlParameter.put("action", "ObjProps");
    urlParameter.put("class",
        "customization.SiteChange2.SiteChangeRequest");
    URL url =
        GatewayURL.buildAuthenticatedURL(wt.enterprise.URLProcessor,
            URLTemplateAction, urlParameter);
    WTContext context = WTContext.getContext();
    context.showDocument(url, "ChangeManager");
}
catch (Exception e) {
    throw new TaskDelegateException(e);
}
```

You will also need to put the following in the `user.imports` section:

```
import java.net.URL;
import java.util.Properties;
import wt.httpgw.GatewayURL;
import wt.util.WTContext
```

4. For `SiteCreateChangeRequestDelegate`:

- The `createChangeItem` should be as follows:

```
Properties props = getFormData(); String name =
props.getProperty("name");
setContextObj(SiteChangeRequest.newSiteChangeRequest(name))
```

- The method `setAttributes` calls `super.setAttributes` in `CreateChangeRequestDelegate` and then sets the justification attribute.

```
super.setAttributes();
Properties props = getFormData();
String justification = props.getProperty("justification");
change_request = (SiteChangeRequest)getContextObj();
change_request.setJustification(justification);
setContextObj(change_request);
```

You will also have to add the following to the `user.imports` section:

```
import customization.SiteChange2.SiteChangeRequest;
import java.util.Properties;
```

Add the following to the `user.attributes` section:

```
private SiteChangeRequest change_request = null;
```

For SiteUpdateChangeRequestDelegate, the updateAttributes method calls the super.updateAttributes in UpdateChangeRequestDelegate and then sets the justification attribute as follows:

```
super.updateAttributes();
props = getFormData();
SiteChangeRequest site_change_request =
    (SiteChangeRequest)getContextObj();
String justification = props.getProperty("justification").trim();
String the_justification = "";
if (site_change_request.getJustification() != null) {
    the_justification = site_change_request.getJustification();
}
if (!(the_justification.equals(justification))) {
    site_change_request.setJustification( justification);
}
```

5. Add any messages or customizations to SiteChange2Resource.java, in particular, a display name for the new attribute and the localized error message which you referenced in the checkAttributes method of SiteChangeRequest.java:

```
({"SiteChangeRequest.justification.DisplayName",
  "Justification"}),
```

Possibly one for the new class:

```
({"SiteChangeRequest.DisplayName", "Site Change Request"})
```

And an error message when the new attribute is missing:

```
({NO_CR_JUSTIFICATION, "A justification must be supplied for
  the change request"})
```

**Note:** If you copied this file from change2Resource.java, be sure to change the package name to customization.SiteChange2 and the class name to SiteChange2Resource.

## Text Editor Tasks

1. Create a SiteChange2.properties file and add a reference to it (as shown in bold below) from wt.properties, as follows:

```
wt.services.applicationcontext.WTServiceProviderFromProperties.  
  customPropertyFiles=htmltemplate.properties,  
  htmlcomponent.properties, wt/change2/change2.properties,  
  customization/SiteChange2/SiteChange2.properties
```

The key is to add the following line:

```
, customization/SiteChange2/SiteChange2.properties
```

to the end of the existing line in wt.properties. This line lists all the properties files that are included in the main properties file and you should not add or remove any others except SiteChange2.properties.

2. Copy ChangeManagementHome.html from /Windchill/codebase/templates/change2, naming the copy SiteChangeManagementHome.html, so that when Create Change Request is selected, a SiteChangeRequest is created rather than a WTChangeRequest2.
  - Change the call to create the URL in the HTML file so a SiteChangeRequest is created rather than a WTChangeRequest2.
  - Add the following line to SiteChange2.Properties so that we find the new home page for change management:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/  
  ChangeManagementHome/java.lang.Object/  
  0=customization.SiteChange2.clients.SiteChangeManagementHome
```

3. Copy HTML CreateChangeRequest.html, UpdateChangeRequest.html, and ViewChangeRequest.html from /Windchill/codebase/templates/change2 (name the copies SiteCreateChangeRequest.html, SiteUpdateChangeRequest.html, and SiteViewChangeRequest.html) and add new HTML as needed. These will be the HTML templates used to display your SiteChangeRequest objects when they are created, updated, or viewed.
  - Add labels and fields for the new attribute named "justification" by copying the HTML for an existing attribute ("description") and pasting it where you want the new attribute to go (just after "description"). Then edit it as necessary to refer to "justification" instead of "description".
  - Also in SiteCreateChangeRequest.html, change the getResourceString Windchill script call getResourceString as follows:

```
getResourceString textToken=  
  "SiteChangeRequest.justification.DisplayName"  
  textResourceBundle=customization.SiteChange2.SiteChange2Resource
```

This gets the label for "justificaton" from your new resource file.

- Change `wt.change2.WTChangeRequest2` to `customization.SiteChange2.SiteChangeRequest` in the `getAcceptLink` Windchill script call in `SiteCreateChangeRequest.html`. This causes the creation of a `SiteChangeRequest` object instead of a `WTChangeRequest2` object. If you forget to do this, then after creating a change request, the view change request page will show a `WTChangeRequest2` object that has no "justification" field.
- Add the following three lines to the `SiteChange2.properties` file to cause the three HTML files to be used with a `SiteChangeRequest` object:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
  ViewChangeObject/customization.SiteChange2.SiteChangeRequest/
  0=customization.SiteChange2.clients.SiteViewChangeRequest
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
  UpdateChangeObject/customization.SiteChange2.SiteChangeRequest/
  0=customization.SiteChange2.clients.SiteUpdateChangeRequest
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
  CreateChangeObject/customization.SiteChange2.SiteChangeRequest/
  0=customization.SiteChange2.clients.SiteCreateChangeRequest
```

Now when Windchill looks for a `DefaultHTMLTemplate` with a `CreateChangeObject` action for a `SiteChangeRequest` object, it will use the `SiteCreateChangeRequest.html` file.

4. Add the following two lines to the `SiteChange2.properties` file to cause the `SiteCreateChangeRequestDelegate` and `SiteUpdateChangeRequestDelegate` delegates you just created to be used with a `SiteChangeRequest` object:

```
wt.services/svc/default/wt.templateutil.processor.FormTaskDelegate/
  UpdateChangeObject/customization.SiteChange2.SiteChangeRequest/
  0=customization.SiteChange2.htmlclient.SiteUpdateChangeRequestDelegate/
  duplicate
wt.services/svc/default/wt.templateutil.processor.FormTaskDelegate/
  CreateChangeObject/customization.SiteChange2.SiteChangeRequest/
  0=customization.SiteChange2.htmlclient.SiteCreateChangeRequestDelegate/
  duplicate
```

## Change Management Delegates

The `wt.change2` package has several delegates defined to allow customization. The following is an overview to help you know where to start.

### ChooseLifeCycleDelegate

`ChooseLifeCycleDelegate` is used within `StandardChangeService2` by calling the protected method `chooseLifeCycle`. In the `chooseLifeCycle` method, the delegate is obtained from the `Change2DelegateFactory`. The `chooseLifeCycle` method is not called if the change object is already persisted, because it would already have a life cycle set. In the `chooseLifeCycle` method, the delegate mechanism is used only if a life cycle has not yet been set.

```
// In any of the saveChange... methods, such as saveChangeRequest:
If changeItem is not persistent {
    ChangeServiceHelper2.service.chooseLifeCycle (changeItem);
}

// In chooseLifeCycle
if changeItem has no lifecycle {
    Set the lifecycle using the delegate;
}
```

When saving a change object, Windchill uses the `ChooseLifeCycleDelegate` mechanism to assign a life cycle if and only if the change object is not yet persistent and a life cycle is not assigned already. This means that the delegate mechanism is effective only when creating a new change object. As a customization, you could allow the user to select a life cycle from a drop-down list in the HTML page and then, because the life cycle would already be set, the delegate would not replace that life cycle.

The `DefaultChooseLifeCycleDelegate` uses a property file entry to get the name of the life cycle that should be used.

### ChooseFolderDelegate

`ChooseFolderDelegate` is used within `StandardChangeService2`. It is obtained from the `Change2DelegateFactory`.

The Windchill vision of change management is that change issues, requests and orders should be visible in folders so that a user can easily look for issues (or suggestions), requests (issues that are being addressed), and orders (changes that have been made). The investigations, proposals, analysis activities, and change activities are tightly tied to other objects and they are not visible in folders. Because these related objects need the same access control and other attributes of their related object, which is in a folder, Windchill puts them in the same cabinet as the related object that is in a folder. The `ChooseFolderDelegate` assigns one `ChangeItem` to a folder/cabinet based on the folder/cabinet of another `ChangeItem`.

The `DefaultChooseFolderDelegate` assigns a `CabinetManaged` object to the same cabinet as a `CabinetManaged` object or a `Folderable` object. It assigns a `Folderable` object to the same folder as another `Folderable` object. It throws an exception when asked to assign a `Folderable` object based on a `CabinetManaged` object.

In addition, a listener is defined on the `change2` service that listens for movement of change issues, request, and orders. It moves their child objects (analysis activities, change activities, change investigations, and change proposals) to the same cabinet into which the parent object was just moved.

## ConcreteAssociationDelegate

`ConcreteAssociationDelegate` is used within `StandardChangeService2`. It is obtained from the `Change2DelegateFactory`.

There are many subclasses of `ConcreteAssociationDelegate`. Each one takes two arguments: the two objects being linked. Each one returns the link object that links the two objects. There is no mechanism for adding additional values for attributes that belong on the link object, so this mechanism works best for link objects that have not been customized with additional attributes.

One other complication with this delegate is that if you customize some of the main classes, you may have to add properties file entries that may not seem intuitively obvious.

Assume that you customize change order (that is, make a subclass of `wt.change2.WTChangeOrder2`) in your `myChange2` package within your customization package and call it `MyChangeOrder`.

When looking up the appropriate subclass of `wt.change2.ConcreteAssociationDelegate`, the following entry in section 9 of `wt.change2.change2.properties` is being used when linking the change order to a change request:

```
wt.services/svc/default/wt.change2.ConcreteAssociationDelegate/  
  wt.change2.WTChangeOrder2/wt.change2.WTChangeRequest2/  
  l=wt.change2.AddressedBy2Delegate/singleton
```

The reference to `wt.change2.WTChangeRequest2` is in a field where inheritance is applied. In other words, if you subclassed `wt.change2.WTChangeRequest2` as `customization.myChange2.MyChangeRequest`, then the delegate lookup process could still use this line in the properties file, because `customization.myChange2.MyChangeRequest` is a `wt.change2.WTChangeRequest2`.

The reference to `wt.change2.WTChangeOrder2` is in a field where inheritance is not applied. In other words, if you subclassed `wt.change2.WTChangeOrder2` as `customization.myChange2.MyChangeOrder`, then the delegate lookup process could not use this line in the properties file because the string "`customization.myChange2.MyChangeOrder2`" does not match exactly the string "`wt.change2.WTChangeOrder2`".

All that is needed is another line in the properties file:

```
wt.services/svc/default/wt.change2.ConcreteAssociationDelegate/  
  customization.myChange2.MyChangeOrder/  
  wt.change2.WTChangeRequest2/  
  l=wt.change2.AddressedBy2Delegate/singleton
```

## **DisplayIdentificationDelegate**

DisplayIdentificationDelegate is used within StandardChangeService2. It is obtained from the Change2DelegateFactory. For further information, see the section on implementing new display identification delegates in the identity service description in the *Windchill Application Developer's Guide*.



# 11

## Customizing Foundation Applications

This chapter describes how to customize the Foundation applications Document Management, Life Cycle, and Workflow.

<b>Topic</b>	<b>Page</b>
Customizing Document Management .....	11-2
Customizing Life Cycle.....	11-14
Customizing Workflow .....	11-16

# Customizing Document Management

## Customizing the HTML Client

The Document Management HTML client consists of sets of templates for the actions Create, Update, and CheckIn. These sets are as follows:

### Create

- CreateWTDocumentGeneralTab.html
- CreateWTDocumentReferencesTab.html
- CreateWTDocumentStructureTab.html
- CreateWTDocumentProcessing.html

### Update

- UpdateWTDocumentGeneralTab.html
- UpdateWTDocumentReferencesTab.html
- UpdateWTDocumentStructureTab.html
- UpdateWTDocumentProcessing.html

### Checkin

- CheckInWTDocument.html
- FirstTimeCheckInWTDocument.html
- CheckInWTDocumentProcessing.html

Each template contains a WTDocumentApplet and form fields for all the document information editable during the operation. The applet and the form fields may be visible on one template and hidden on another.

For example, the **General** tab displays the WTDocumentApplet and the form fields for general document information like name, number, and so on but hides the fields with information about document references and structure. The **Structure** tab displays the document structure but hides the WTDocumentApplet (width and height of 0), the general document information fields, and document references. All fields on the **Processing** page are hidden. The value for each field is set by a Windchill script call to contextualValue for that field's name.

## Adding an Attribute to a Template

To add an attribute to a template, the form field must be added to every template in the set, including the Processing template. On the template where the attribute is to be entered, that form field must be one of the visible types of HTML form controls (for example, text, textarea, menu, and so on). On the other templates, where that attribute is not intended to be visible, the form field should be hidden HTML input. On template sets where a particular field will be display-only, it should be displayed as text rather than as a form field (see the Update action's **General** tab, for example).

The name of the new attribute can either be added to a resource bundle and accessed via a Windchill script call to getResourceString (the recommended method), or hardcoded directly into the template.

The following is an example of a new attribute using an excerpt from the Create action's **General** tab:

```

<TR>
  <TD colspan=2>

    <APPLET name=formApplet code="wt/boot/BootstrapApplet.class"
      width=600 height=80 archive="wt/security/security.jar">
      <PARAM name="boot_jar" value="docmgr.jar">
      <PARAM name="boot_class" value="wt.clients.doc.WTDocumentApplet">
      . . .
    </APPLET>

    . . .
    <!-- add hidden fields from other tabs here -->

    . . .

    <!-- hidden fields holding values for creating structure links -->
    <INPUT name = "removeStructure" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
      propertyName=removeStructure</SCRIPT>">
    <INPUT name = "structure" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
      propertyName=structure</SCRIPT>">

    </TD>
  </TR>
  <TR>
    <TD align=right>
      <B><FONT FACE=arial,Helvetica><<SCRIPT LANGUAGE=
        Windchill>getResourceString textToken="docNumberLbl"
        textResourceBundle=wt.clients.doc.DocRB</SCRIPT></FONT></B>
    </TD>
    <TD>
      <INPUT name = "Number" type="text" value=
        "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=Number</SCRIPT>">
    </TD>
  </TR>
  <TR>
    <TD align=right>
      <B><FONT FACE=arial,Helvetica><<SCRIPT LANGUAGE=
        Windchill>getResourceString textToken="docNameLbl"
        textResourceBundle=wt.clients.doc.DocRB</SCRIPT></FONT></B>
    </TD>
    <TD>
      <INPUT name = "Name" type="text" size=50 value=
        "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=Name</SCRIPT>">
    </TD>
  </TR>
  <TR>
    <TD align=right>
      <B><FONT FACE=arial,Helvetica><<SCRIPT LANGUAGE=
        Windchill>getResourceString textToken="newAttributeLbl"
        textResourceBundle=myPackage.myRB</SCRIPT></FONT></B>

```

```

</TD>
<TD>
  <INPUT name = "newAttribute" type="text" size=50 value=
    "<SCRIPT LANGUAGE=Windchill>contextualValue
      propertyName=newAttribute</SCRIPT>">
</TD>
</TR>
. . .

```

The following is an example of a new attribute using an excerpt from the Create action's **Structure** tab:

```

TR>
  <TD>
    <APPLET name=formApplet code="wt/boot/BootstrapApplet.class"
      width=0 height=0 archive="wt/security/security.jar">
      <PARAM name="boot_jar" value="docmgr.jar">
      <PARAM name="boot_class" value="wt.clients.doc.WTDocumentApplet">
    . . .
    </APPLET>
    . . .
    <!-- hidden fields that hold values from General tab -->
    . . .
    <INPUT name = "Number" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=Number</SCRIPT>">
    <INPUT name = "Name" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=Name</SCRIPT>">
    <INPUT name = "newAttribute" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=newAttribute</SCRIPT>">
    <!-- add hidden fields from other tabs here -->
    <!-- hidden fields holding values for creating dependsOn links -->
    <INPUT name = "removeDependency" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=removeDependency</SCRIPT>">
    <INPUT name = "removeComment" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=removeComment</SCRIPT>">
    <INPUT name = "dependsOn" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=dependsOn</SCRIPT>">
    <INPUT name = "dependencyComment" type="hidden" value=
      "<SCRIPT LANGUAGE=Windchill>contextualValue
        propertyName=dependencyComment</SCRIPT>">

```

## Processing a New Attribute

When you click **OK**, the value of the new attribute is collected, along with the other attributes, and placed in a `wt.doc.DocumentFormData` object (a wrapper of a `Hashtable`). This object is passed into the `wt.doc.WTDocumentDelegate` for processing (see the section titled `Customizing Structure and References Processing` later in this section). To process the new attribute, `WTDocumentDelegate` must be subclassed and the appropriate methods must be overridden to add processing for the new attribute. You must definitely override `createDocument`. You must override `updateDocument` or `checkinDocument` if you are allowing modification of the new attribute during update or checkin. The new attribute can be accessed using the name of its form field, for example:

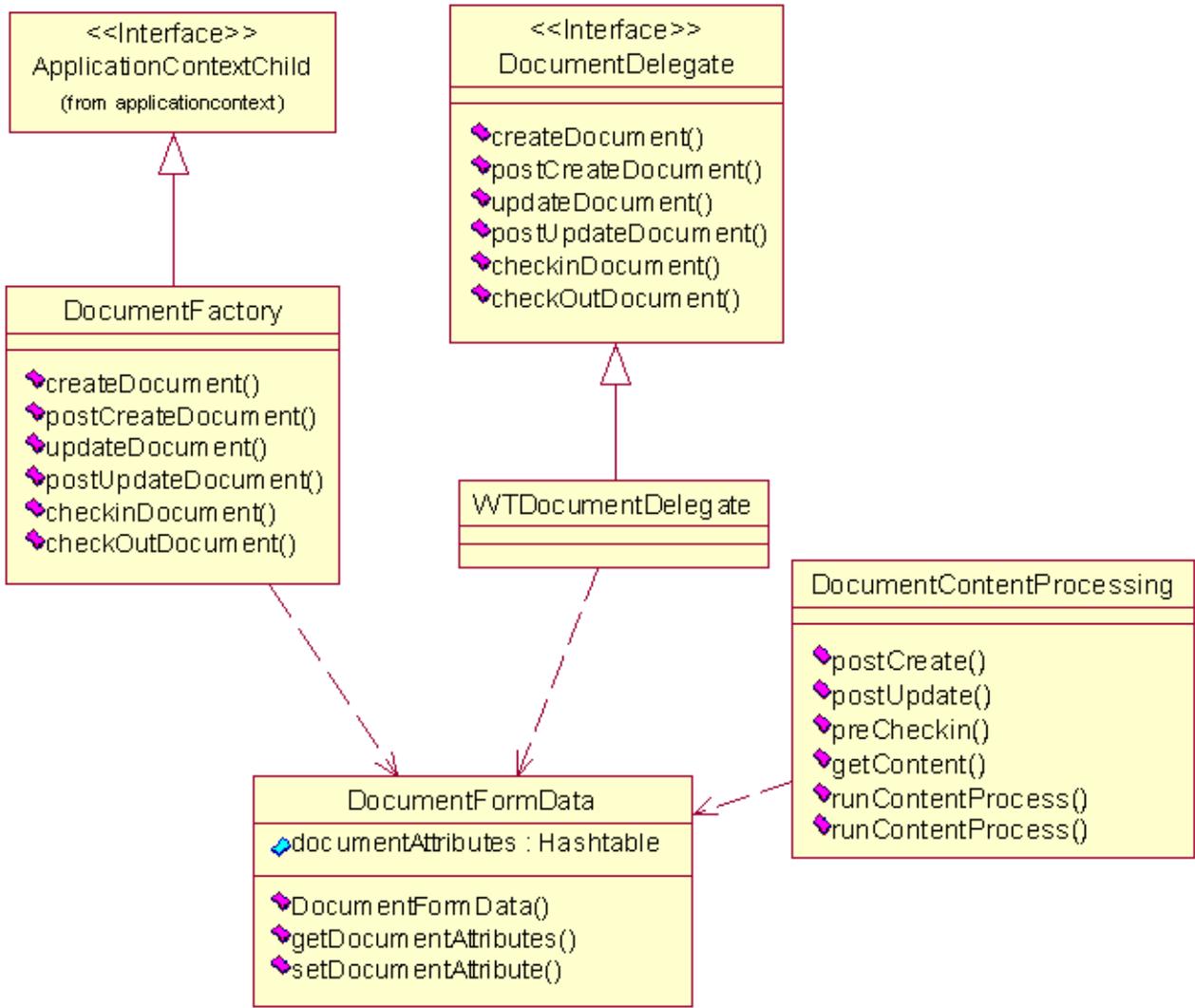
```
form_data.getDocumentAttributes( "newAttributeName" );
```

## Customizing Structure and References Processing

The out-of-the-box client provides very simple manual processing for the binary link classes and relationships provided with documents. The Java client uses the association bean `wt.clients.beans.AssociationsPanel` to create and update both the structure and references links. This panel does not offer any customization points.

In the HTML client, however, customization points were put in the system to allow customer sites to automate some of their specific processing of document structure and references. The `Get Content` and `Checkout` functions in the Java client offer the same customization points that are available in the HTML client for the downloading of files.

Some operations are best performed on the server because they involve reading and writing to the database. The server-side customization points are located in the wt.doc.WTDocumentDelegate class (shown in the following figure), which uses a factory to instantiate it so it can be replaced by a customized version.



The source for `wt.doc.WTDocumentDelegate` is provided as an example of what must be executed in the methods. The customized code can be added in two ways: by subclassing either `wt.doc.DocumentDelegate` or `wt.doc.WTDocumentDelegate`. If `wt.doc.DocumentDelegate` is subclassed, all methods must be implemented in the new class. The entry in `service.properties` must be modified or an additional entry added if a subclass of `wt.doc.WTDocument` is to be processed differently. An example entry follows:

```
wt.services/svc/default/wt.doc.DocumentDelegate/null/  
wt.doc.WTDocument/0=wt.doc.WTDocumentDelegate/duplicate
```

There are also operations that must be performed on the client because they involve out-of-the-sandbox processing. They must be executed on the client where the content or files are to be uploaded from or downloaded to. The client-side customization points are put in `wt.doc.DocumentContentProcessing` (shown in the preceding figure), which is also provided as source. The methods in `DocumentContentProcessing` are called through a combination of a property in `wt.properties` and reflection so they can be changed to a custom method easily. The property in `wt.properties` currently defaults to the following setting:

```
wt.doc.contentProcessingDelegate=wt.doc.DocumentContentProcessing
```

Subclass the class `wt.doc.DocumentContentProcessing` to override the specific methods of interest. Then change the `wt.doc.contentProcessingDelegate` setting to the new subclass. But, unlike the `service.properties` version of delegation, this allows only one `DocumentContentProcessing`, not one per class.

## Customizing Create Document

There are three customization points in the create document processing in the HTML client. The applet embedded in the HTML forms is used to process the various steps of the create. The basic steps are as follows:

1. Create the document calling the method `createDocument` in `wt.doc.WTDocumentDelegate` through a call to the document helper, for example:

```
document = WTDocumentHelper.service.createDocument("", form_data);
```

2. Save the primary content using the primary content bean (`wt.clients.util.http.HTTPUploadDownloadPanel`). This is not a customization point.
3. Create the relationships and start the life cycle processing using `postCreateDocument` in `WTDocumentDelegate`, which is also called through the document helper.

4. Allow for any create content postprocessing by calling `DocumentContentProcessing` to initiate a post-create method. This call looks like the following:

```
String message = DocumentContentProcessing.runContentProcess(  
    DocumentContentProcessing.POSTCREATE,  
    form_data,documentHandle);
```

Steps 1, 3, and 4 are the customization points that can be modified by overriding the class or method listed. In Step 4, the method that is called by `POSTCREATE` by default is `DocumentContentProcessing.postCreate`. This is currently a stub method that was created to allow customer sites to do content processing on the client, other than the primary content of the document being created. The primary content for the document being created is handled through the primary content bean so no customization is allowed. In Steps 1 and 3, the `WTDocumentDelegate.createDocument` and `WTDocumentDelegate.postCreateDocument` are not stub methods but they can be overridden, as described earlier, to add additional customized processing on the server.

The `wt.doc.DocumentFormData` class is a wrapper of a `Hashtable` that is used to pass data between the methods. Any fields added to the hidden fields of the three HTML files that make up the create forms and the create processing file should be available from the `DocumentFormData` instance passed into these methods (see the section titled `Customizing the HTML Client` earlier in this section for information on customizing these forms). To see the values that are available, use the `DocumentFormData.getDocumentAttributes` method to get the entire `Hashtable` to output the available values.

## Customizing Update Document

The update customization points are very similar to the create document customization points. The applet performs the following basic steps:

1. Update the basic metadata by calling the `updateDocument` method in `wt.doc.WTDocumentDelegate` through a call to the document helper.
2. If the checksums value on the file listed on the update form is different than that of the file saved in the database, the primary content bean is called to upload the file. This is not a customization point.
3. Update the relationships by calling the `postUpdateDocument` method in `wt.doc.WTDocumentDelegate` through a call to the document helper.
4. Allow for any update content postprocessing by calling `DocumentContentProcessing` to initiate a post-update method. Currently, `wt.doc.DocumentContentProcessing.postUpdate` is the stub method that is called.

## Customizing Check Out Document

The checkout of a document has two customization points:

- `wt.enterprise.CheckOutLFHDelegate` is run for the checkout processing of the document from the HTML client. The checkout is run by calling the `checkOutDocument` method in `wt.doc.WTDocumentDelegate` through a call to the document helper.
- The applet that is embedded to do the local file handling eventually calls `wt.doc.DocumentContentProcessing.getContent`. For checkout, the location to which the primary content is downloaded is remembered in the database so that on an update or checkin of this document, the location of the file defaults to the checkout location.

The same file handling code is called in the Java client and the HTML client for the local file handling. Therefore, any custom code added for the `getContent` method called by `wt.doc.DocumentContentProcessing` is also executed in the Java client. The download of the primary file for the document being checked out is not a customization point. The `getContent` method gets a configuration specification and uses it to navigate through the structure. To change which configuration specification is used to download the file, subclass `wt.doc.DocumentContentProcessing` and override `downloadStructureContent` and `getContent`.

There is one configuration choice on how the download of the document is performed that does not involve changing or adding code. If there is structure associated with the document and a primary file on the document (not a URL), the following property in `wt.properties` is used to determine if the tree of the document structure is navigated and all of the primary files for those documents are downloaded:

```
wt.doc.structureDownload=true
```

By default, the property is true if it is not set in `wt.properties`, and all of the primary content from the structure is downloaded. This property also applies to both Java and HTML clients, and Get Content Java and HTML.

## Customizing Check In Document

There are two customization points for checkin. The applet performs the following basic steps:

1. If the checksums value on the file listed on the checkin form is different than that of the file saved in the database, the primary content bean is called to upload the file. This is not a customization point.
2. Allow for any checkin content pre-processing by calling `wt.doc.DocumentContentProcessing` to initiate a pre-checkin method. Currently, `wt.doc.DocumentContentProcessing.preCheckin` is the stub method that is called.
3. Perform the checkin on the document by calling the `checkinDocument` method in `wt.doc.WTDocumentDelegate` through a call to the document helper.

## Customizing Get Content

Because get content is a client-side function, there are no server-side processing customization points. Get content does not modify any data in the database. It downloads only the primary file (and primary files of the structure). The applet that is embedded in the HTML client and the download used in the Java client both call the same method that is used in the checkout document, the `getContent` method called by `wt.doc.DocumentContentProcessing`. By default this method is `wt.doc.DocumentContentProcessing.getContent`. See the preceding section titled Customizing Check Out Document for further information on modifying `wt.doc.DocumentContentProcessing.getContent`.

## Customizing Create Document Template for a WTdocument Subtype

The following steps are needed to create a document template for a `WTDocument` subtype. Consider that a new `WTDocument` subtype has been created by name `MyWTDocument` and that a few new attributes have been added to it.

1. The first step is a customization point that is required to initialize an object of a `WTDocument` subtype.

### Extend

`com.ptc.windchill.enterprise.templates.doc.server.TemplatesWTDocumentDelegate` to create a new delegate for `MyWTDocument` that will be called `TemplatesMyWTDocument`. (This will require a rose model change.) The `newTemplate` method in `TemplatesMyWTDocument` needs to be overridden.

Rose does not generate a `newTemplate` method in `TemplatesMyWTDocument`. A `newTemplate` method with the exact same signature as in `TemplatesWTDocumentDelegate` needs to be added to the

new delegate, either in the TemplatesMyWTDocument java file under the user operations section, or directly in rose.

The overridden newTemplate method in TemplatesMyWTDocument should look like this

```
public Templateable newTemplate( String objType, String context
)
    throws WTEException {
    MyWTDocument template = null;
    template = WTDocument.newMyWTDocument();
    template = initNewTemplate(template , objType, context);
    return template;
}
```

In order for the methods in the new delegate class to be called while creating a document template for a MyWTDocument modeled subtype, a new property entry needs to be added to codebase\com\ptc\windchill\enterprise\EnterpriseServerFactoryDelegate.properties which should look like this (needs to be on one line) -

```
wt.services/svc/default/com.ptc.windchill.enterprise.templates.TemplatesObjectDelegate/null/wt.doc.MyWTDocument/0=com.ptc.windchill.enterprise.templates.doc.server.TemplatesMyWTDocumentDelegate/duplicate
```

2. After the above preliminary customization, create a document template for a modeled subtype of WTdocument as follows:

- Call the newTemplate delegate method for a MyWTDocument subtype with the obj\_type parameter set to wt.doc.MyWTDocument and context set to 'Classic' for WindchillPDM

```
MyWTDocument doc = null;
```

```
doc = (MyWTDocument)TemplatesFactory.newTemplate(obj_type, context);
```

- Set the user selected attributes on the document template such as the name, number, title, description, organization, enabled flag, location, lifecycletemplate, teamtemplate, department, document type and any other new attributes defined for the MyWTDocument.
- Follow these steps to perform some solution specific tasks for a MyWTDocument object, during the create operation

The context passed in here should be 'Classic' for WindchillPDM

```

HashMap mapCreate = new HashMap();

mapCreate.put("context", context);

doc =
(MyWTDocument)TemplatesFactory.doCreateTemplateContextTask
s("Enterprise", doc, doc, mapCreate);

```

- Persist the template document object

```

if (doc != null)

doc = (MyWTDocument)PersistenceHelper.manager.save(doc);

```

## Customizing Create Document from Template from a WTDocument Subtype

The following steps are needed to create a document from a template that is of a WTDocument subtype. Consider that a new WTDocument subtype has been created by name MyWTDocument and that a template has been created with this type.

1. The customization point here is to first create a new copy delegate for a MyWTDocument subtype using the steps already described under “Adding ‘Save As’ functionality to a new Document or part Class”.
2. After the above preliminary customization, create a document from a WTDocument subtype template as follows.

- Call the newObjFromTemplate delegate method with the MyWTDocument subtype template to create the document from and the context set to ‘Classic’ for WindchillPDM

```

MyWTDocument doc = null;

doc =
(WTDocument)TemplatesFactory.newObjFromTemplate(template,
context);

```

- Set the user selected attributes on the document such as the name, number, title, description, organization, location, lifecycletemplate, teamtemplate, department, document type and any other new attributes defined for the MyWTDocument.
- Persist the document object created from template using the copy service, passing in the template and the document to be created from it.doc =

```

EnterpriseHelper.service.saveCopy((RevisionControlled)template,doc);

```
- Follow these steps to perform some solution specific tasks for a MyWTDocument object, post create operation, on an as required basis.

If 'submit to lifecycle' capability is required after creating a document

from template, set the 'submit' key in the HashMap to the user selected value.

If the 'keep checkout' capability is required on the document created

from template, set the 'keepCheckedOut' key in the HashMap to the user selected value.

```
HashMap mapPCreate = new HashMap();
```

```
mapPCreate.put("submit", submit);
```

```
mapPCreate.put("keepCheckedOut", keepCheckedOut);
```

```
wtdoc =
```

```
(WTDocument)TemplatesFactory.doPostCreateFromTemplateContextTasks("Enterprise", wtdoc, wtdoc, template, mapPCreate);
```

## Customizing Life Cycle

The way life cycle state information is displayed, particularly in life cycle-managed object properties pages, can be customized by setting configurable components in the `wt.lifecycle.lifecycleResource*.java` files. You can set values for the following entries:

### **STATES\_SEPARATOR**

When displaying a string of states, this value is the separator between each of the states listed. Default is " - ".

### **CURRENT\_STATE\_BEGIN**

When displaying a string of states, this value is the notation that a particular state is the current state. Default is "<B>".

### **CURRENT\_STATE\_END**

When displaying a string of states, this value is the notation that a particular state is the current state. Default is "</B>".

### **DROPPED\_STATE\_BEGIN**

This entry notes that the current state (usually dropped) is not found in the list of current states. Default is " [".

### **DROPPED\_STATE\_END**

This entry notes that the current state (usually dropped) is not found in the list of current states. Default is "] ".

### **IS\_AT\_GATE**

This value is used to indicate that Awaiting Promotion = true. Default is "Yes".

### **IS\_NOT\_AT\_GATE**

This value is used to indicate that Awaiting Promotion = false. Default is "No".

### **LABEL\_BEGIN**

This value is used when displaying any of the StateProcessor labels. Default is "<B>".

### **LABEL\_END**

This value is used when displaying any of the StateProcessor labels. Default is ": </B>".

### **STATE\_LIST\_BEGIN**

When the list of states is provided along with other information, this entry differentiates the State list from other information in the display. Default is "(".

### **STATE\_LIST\_END**

When the list of states is provided along with other information, this entry differentiates the State list from other information in the display. Default is ")".

### **HISTORY\_LABEL**

This entry is used when displaying a link to the Life Cycle History page. Default is "History".

**HISTORY\_NOTATION\_BEGIN**

When the history link is provided along with other information, this entry differentiates the History link from other information in the display. Default is " (".

**HISTORY\_NOTATION\_END**

When the history link is provided along with other information, this entry differentiates the History link from other information in the display. Default is ")".

# Customizing Workflow

This section describes how to customize HTML templates for workflow activities, and Change Management workflow process templates.

## Customizing Workflow HTML Templates

This section describes how to customize HTML templates for workflow activities. Following are possible ways you could use this capability:

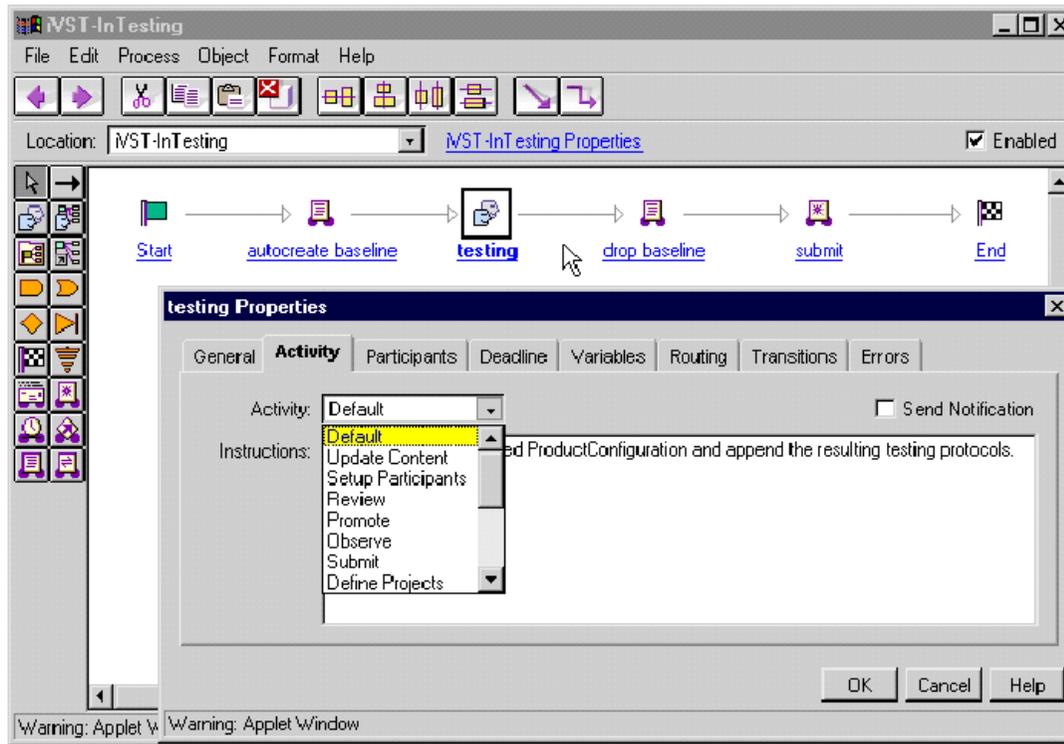
- To change the layout of the standard workflow activity property pages to add customer logos and so forth.
- To add read-only displays of process variables.
- To integrate information (attributes, icon, associations, and so on) of the primary business object of the workflow.
- To display a summary of process variables, for example, the voting and comments of different reviewers as the basis for a promote decision.

The functionality to perform all of these customizations is available through existing Windchill scripts, as seen in the standard Windchill HTML client. As a result, you can make these customizations without programming Java (possibly only one resource bundle change), but only editing HTML and some entries in a Windchill properties file.

For both an overview and detailed information about customizing HTML clients, see Chapter 7, Customizing the HTML Client. This section discusses the HTML client only as it applies to workflow.

To customize a workflow HTML template, perform the following steps:

1. Create a workflow activity that you want to be represented by the customized HTML page with the workflow administrator applet. The action argument of the activities property page URL is specified by the type of workflow task (see the following figure).



For customizations, you can use the predefined HTML templates UserTask1.html through UserTask7.html, or create your own named templates, for example CustomTask.html. If you create your own task template, you must add an entry to the wt.properties file, as shown in the following example, to cause the new task to appear in the list of available tasks:

```
wt.clients.workflow.tasks.task.1=WfTask
wt.clients.workflow.tasks.task.2=WfUpdateContent
wt.clients.workflow.tasks.task.3=WfAugment
wt.clients.workflow.tasks.task.4=review
wt.clients.workflow.tasks.task.5=promote
wt.clients.workflow.tasks.task.6=observe
wt.clients.workflow.tasks.task.7=submit
wt.clients.workflow.tasks.task.8=WfDefineProjects
wt.clients.workflow.tasks.task.9=WfChgMgmt
wt.clients.workflow.tasks.task.10=UserTask1
wt.clients.workflow.tasks.task.11=UserTask2
wt.clients.workflow.tasks.task.12=UserTask3
wt.clients.workflow.tasks.task.13=UserTask4
```

```
wt.clients.workflow.tasks.task.14=UserTask5
wt.clients.workflow.tasks.task.15=UserTask6
wt.clients.workflow.tasks.task.16=UserTask7
wt.clients.workflow.tasks.task.17=CustomTask
```

The default behavior of workflow task processing recognizes these tasks by their template name, for example, CustomTask. If you want to change the display name of your task, you can add an entry to the attribute declaration and the Tasks section of the wt.clients.workflow.tasks.TasksRB.java file and compile it. The following is an example of such an addition to the attribute declaration:

```
public static final String MYCUSTOMIZEDTASK= "CustomTask";
```

The following is an example of such an addition to the Tasks section:

```
{MYCUSTOMIZEDTASK, "Customized Task Name"},
```

The string "Customized Task Name" is displayed in the drop-down list and must be localized. The string "CustomTask" is the value of the action argument in the URL requests of the HTML page for a workflow activity of that type.

2. Add or modify the appropriate entries in the service.properties and htmltemplate.properties files. If you add an entry in the resource bundle, you must add a line similar to the following in the service.properties file:

```
wt.services/svc/default/wt.enterprise.TemplateProcessor/
  CustomTask/java.lang.Object/
  0=wt.workflow.worklist.WfTaskProcessor/
  duplicate
```

You must also add a line similar to the following in the htmltemplate.properties file:

```
wt.services/rsc/default/wt.templateutil.DefaultHTMLTemplate/
  CustomTask/java.lang.Object/0=
  customizations.workflow.templates.CustomTask
```

3. Create or modify the HTML template file (customizations.workflow.templates.CustomTask). For examples, see the standard HTML template files listed in the htmltemplate.properties file. The Windchill scripts you can use are defined primarily by the wt.workflow.worklist.WfTaskProcessor and the wt.enterprise.BasicTemplateProcessor (see the Javadoc for further information).

The following are examples of useful functionality from the more commonly used Windchill scripts:

– From `wt.workflow.worklist.WFTaskProcessor`:

- To print workflow variables, use the following scripts:

```
activityVariable variable=<Yourvariablename> rows=# columns=#
processVariable variable=<Yourvariablename> rows=# columns=#
```

- To print information about the Process, Activity, and PrimaryBusinessObject in the same page, use the following script:

```
setContextObject context=process | activity|
primaryBusinessObject
```

This script does not print any HTML code but switches the ContextObject of the TemplateProcessor. This means the scripts that follow this script in the HTML page can generate information about the new ContextObject (for example, attribute names and values of the primaryBusinessObject or a table of associated objects).

– From `wt.enterprise.BasicTemplateProcessor`:

- To create a link to another dynamically generated page, use the following script (see the Javadoc for further information):

```
objectActionLink action="action" [label="label" ]
labelPropertyName="labelPropertyName" ]
```

- To display the label and value of attributes of a Windchill object, enter the following scripts:

```
objectPropertyName propertyName="propertyName"
objectPropertyValue propertyName="propertyName"
```

Attributes can be "name", "number", "checkoutInfo", "location", and so on.

- To display a subtemplate (for example, to display a usedBy-table), enter the following script:

```
processSubTemplate action="action" [SubTemplate depending
name/value pairs]
```

Most of the time you will have to add entries in the properties files to create new actions and new related HTML template files that hold only a part of an HTML page (for example, a single table).

# Customizing Change Management Workflow Process Templates

## Introduction

Synchronization robots are a critical part of the Change Management workflow process template examples. Originally, these robots were developed as pure expression robots. However, use of normal expression synchronization robots causes very heavy Oracle activity, resulting in frequent Oracle redo log turnover. At Release 5.1, therefore, these robots were updated to use object or class event synchronization in conjunction with an expression.

These changes affect only out-of-the-box example workflow process templates in `loadfiles\ChangeManagement.csv`. If you use these templates but have not customized them, simply delete the existing templates and load the modified ones. If you have customized the out-of-the-box workflow process templates, you should manually incorporate the changes, as described in the remainder of this section, into your customizations to take advantage of the improved performance. If your own expression synchronization robots are problematic, you should manually incorporate changes similar to those described in this section.

Following are the Change Management workflow process templates that have been enhanced to improve system performance:

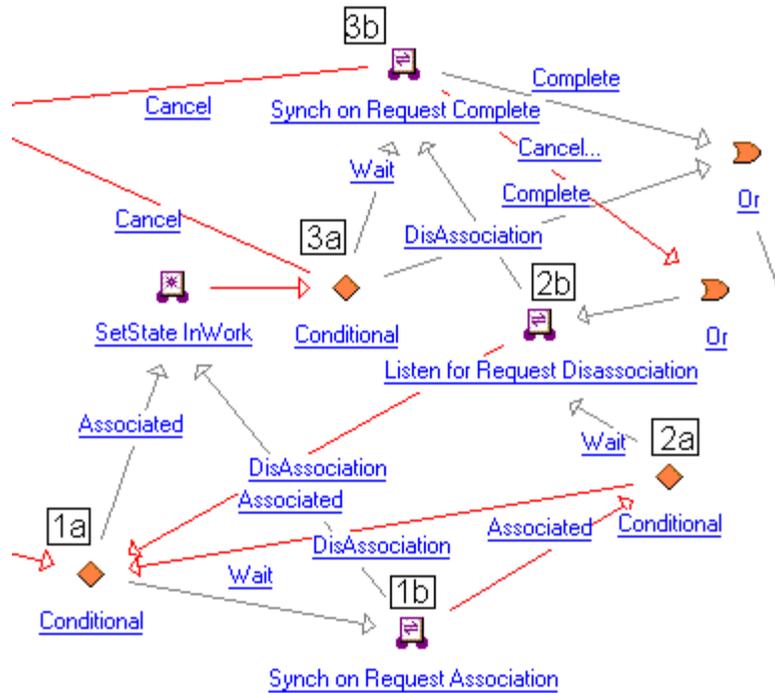
- Change Issue Process
- Change Request Process 2
- Change Investigation Process
- Change Proposal Process
- Change Analysis Process
- Change Order Process
- Change Activity Process

## Change Issue Process

Both of the existing robots in this template have been converted to object event synchronization robots, and a new robot of that type has also been added:

Template	Robot
Change Issue Process	Synch on Request Association
Change Issue Process	Listen for Request Disassociation
Change Issue Process	Synch on Request Complete

The expression logic of change issue robots is illustrated in the following figure.



The following table explains the details of this figure; the numbers in the table correspond to numbers in the figure.

<p>1a) This conditional router checks if the change issue is already associated to a change request. If so, the workflow continues to the conditional at 3a; otherwise, it proceeds to the synch robot at 1b.</p>	
<p>1b) The Synch on Request Association robot waits until the event ISSUE_FORMALIZED is emitted for the primaryBusinessObject (the change issue), signaling the change issue has been attached to a change request. The workflow then continues to the conditional at 2a and the conditional at 3a simultaneously.</p>	
<p>2a) This conditional router checks if the change issue has been immediately disassociated with the change request. If so, the workflow cycles back to 1a; otherwise, it continues to the synch robot at 2b.</p>	<p>3a) This conditional router checks the state of the associated change request. If it is in state Completed, the workflow continues to 3b (the end of the process). If it is in state Cancelled, the workflow loops back to a point near the beginning of the process. Otherwise, the workflow continues to the synch robot at 3b.</p>

<p>2b) The Listen for Request Disassociation synch robot listens for the event <b>ISSUE_UNFORMALIZED</b> on the primaryBusinessObject, signaling the change issue has been disassociated from its change request. This causes the synch robot at 3b (→) to terminate, and the workflow to loop back to the conditional at 1a.</p>	<p>3b) The Synch on Request Complete robot waits until the state of the associated change request changes. If the state is Completed, the workflow continues to 3b (the end of the process). If it is in state Cancelled, the workflow loops back to a point near the beginning of the process. Otherwise, the synch robot continues to wait for one of those two states.</p>
---	---

### Synch on Change Request Submit

Each of the following robots has been converted to an object event synchronization robot as described below:

Template	Robot
Change Investigation Process	Synch on Request Submit
Change Analysis Process	Synch on Request Submit
Change Proposal Process	Synch on Request Submit
Change Order Process	Synch on Request Submit
Change Activity Process	Synch on Request Submit

Before Release 5.1, the expression logic performed the following actions:

1. Determine the parent change request by navigating one or two levels of associations based on the change object:
  - For a change investigation or change proposal, navigate the ResearchedBy association.
  - For a change order, navigate the AddressedBy2 association.
  - For an analysis activity, first navigate the DetailedBy association to obtain a change investigation or change proposal, then navigate the ResearchedBy association.
  - For a change activity, first navigate the IncludedIn2 association to obtain a change order, then navigate the AddressedBy2 association.
2. Determine the current life cycle state of the change request.
3. Determine the current value stored in the Complexity attribute (for Synch on Request Submit only).

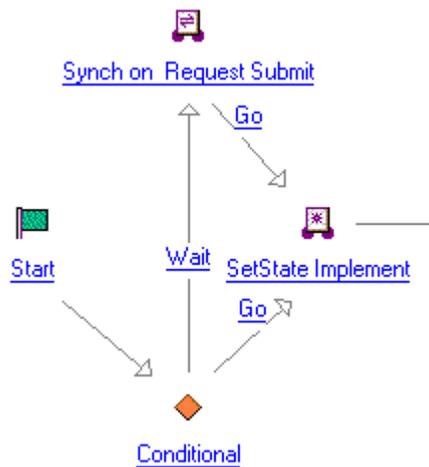
- Based on the results of step 2 and step 3, either continue holding or move on to one of several possible activities that follow in the workflow.

The following changes have been made to this logic:

- A new process variable named `parentChangeRequest` has been introduced in each workflow. It holds the parent request and is initialized when the workflow begins through a transition expression on the process Start transition.
- The following figure shows the expression synchronization robot as it existed before Release 5.1.



This robot has been replaced with a conditional router followed by the synch robot. The new robot, which has been changed to an object event synchronization robot, is shown in the following figure.



The object is the new change request process variable just described. The event is `STATE CHANGE` in each case. The conditional router contains exactly the same logic as the expression in the object event subscription robot. The purpose for this conditional router is to immediately check whether the state has already been reached. This helps avoid the race condition of the state being achieved prior to the instantiation of the synchronization robot. (In both figures, the `parentChangeRequest` variable is assumed to be initialized already.)

- The expression in the object event synchronization robot and conditional has been changed to use the workflow variable `parentChangeRequest` directly, rather than access the database repeatedly to determine the parent change request.

These changes resulted in the following performance improvements:

- Lookup time during each execution of the synchronization expression has been shortened as a result of saving the change request in a workflow variable.
- The synchronization expression executes only after the state of the change request has changed. As a result, there is a very good chance the proper state has been reached each time the expression runs.

## Synch on Multiple Object State Change

Each of the following robots has been converted to a class event synchronization robot as described below.

Template	Robot
Change Request Process 2	Synch on Investigation
Change Request Process 2	Synch on Proposal
Change Request Process 2	Synch on Change Orders
Change Investigation Process	Synch on Analysis Activities
Change Proposal Process	Synch on Activities
Change Order Process	Synch on Change Activities

Before Release 5.1, the expression logic performed the following actions:

1. Determine which children objects are applicable to the synchronization. For example, in the Synch on Change Activities robot, all the change activities related to the change order are relevant.
2. Determine the life cycles states of all the relevant objects.
3. Based on the result of step 2, either continue holding or move on to one of several possible activities that follow in the workflow.

Release 5.1 includes the following changes to this logic:

The expression synchronization robot has been replaced with a conditional router followed by the synch robot. The synch robot has been changed to a new class event synchronization robot. The class differs depending on the particular synchronization robot, but the event is always STATE CHANGE. The conditional router contains exactly the same logic as the expression in the object event subscription robot. The purpose for this conditional router is to immediately check whether the state has already been reached. This helps avoid the race condition of the state being achieved prior to the instantiation of the synchronization robot.

This change resulted in the following improvements:

The synchronization expression is executed only when an object in the proper class has changed state. As a result, the expression is executed only when there is a chance that the states of all related objects are in synchronization.

## Installation and Upgrade

The file `\Windchill\loadfiles\ChangeManagement.csv` contains the definition of the following items:

- ChangeItems Domain
- Example Projects
- Example Workflows
- Example Life Cycle Templates

## New Installations

A new Windchill installation will include the new Change Management workflow process templates. Be sure to load the "Change Management lifecycles and workflows" during the initial database load.

For further information about loading data, see the *Windchill Installation and Configuration Guide*.

## Existing Installations

If you are not concerned about overwriting existing demo workflow process or life cycle templates, you can simply initiate "java wt.load.Demo " and answer "no" to all questions except "Change Management lifecycles and workflows" (see the *Windchill Installation and Configuration Guide* for further information about loading data). You can ignore errors regarding the "Change Items" Domain and the example projects. However, to avoid these errors, remove all sections from `ChangeManagement.csv` except those for the Change Management workflow process templates and life cycle templates.

If you do not want to overwrite the existing demo workflow process templates, PTC recommends that you perform one of the following options before loading the new workflow and life cycle templates:

- Rename the existing workflow and life cycle templates using the Workflow Administrator and Life Cycle Administrator.
- Rename the new workflow and life cycle templates by manually editing the `ChangeManagement.csv` file.

After loading the Release 5.1 workflow process and life cycle templates, you can restore them to their original state one at a time by clicking **Delete Latest Iteration** from the Workflow Administrator or Life Cycle Administrator page.

## Custom Workflow Process Templates

If you have custom workflow process templates that contain synchronization robots, PTC suggests that you load the Release 5.1 example workflow process templates and study how they work, along with this document. This will help you determine if you would benefit from implementing these techniques in your own synchronization robots.

## Code Impacted

The following code has been impacted by the enhancements:

- The workflow processes in loadfiles\ChangeManagement.csv, including the changes described in this section.
- wt.change2.process.ProcessHelper -- A new, overloaded version of the checkRequestFinished method has been added, which takes a change request and checks the state of the passed object.
- wt.change2.StandardChangeService -- The methods saveFormalizedBy and deleteFormalizedBy now emit the events ISSUE\_FORMALIZED and ISSUE\_UNFORMALIZED, respectively.
- wt.admin.AdminEventResource -- The ISSUE\_FORMALIZED and ISSUE\_UNFORMALIZED events were added to this resource bundle (and all of its language variants).
- wt.workflow.robots.synchEventResource -- The ISSUE\_FORMALIZED and ISSUE\_UNFORMALIZED events were added to this resource bundle (and all of its language variants).
- wt.notify.notify.properties -- The ISSUE\_FORMALIZED and ISSUE\_UNFORMALIZED events were added to this property file.

# 12

## Customizing Workgroup Managers

This chapter describes how to enable support for custom parts and how to customize automatic part generation for the Pro/ENGINEER, CATIA, and CADD5 workgroup managers.

<b>Topic</b>	<b>Page</b>
Enabling Support for Custom Parts.....	12-2
Customizing Automatic Part Generation .....	12-5

## Enabling Support for Custom Parts

In the Workgroup Managers for Pro/ENGINEER, CATIA, and CADD5, you can enable support for *custom parts*, which extend wt.part.WTPart. However, a custom part must be modeled before any changes are made to the Workgroup Manager. (For information on extending the Windchill object model, see the *Windchill Application Developer's Guide* and relevant chapters of this manual.)

These Workgroup Managers permit use of custom parts in most operations, including Download, Checkout, Checkin, Associate, Disassociate, Open Structure, and so on. However, the operations used to create parts, New Part and Parts for Documents (automatic part generation), are specific to WTPart. Additionally, when you view the properties of a custom part, new modeled information will not be displayed. The properties page will display any IBAs you may have added to the custom part.

Automatic custom part generation is supported through the **File>Parts for Documents** menu command available within the Workspace Browser. To enable automatic custom part generation when using this command, however, you must either create or modify your *automatic part creator*. For more information, see Automatic Part Generation, later in this chapter.

**Note:** In this section, wt.part.MYWTPart is used as an example of a custom part.

## Modifying File Types

To enable recognition of a custom part in the Workgroup Manager's **Selection Dialog**, modify the preference files used to configure the dialog.

1. In the following preference files, add the class name of the custom part (for example, MYWTPart) to each typelist preference value that contains WTPart. If there is more than one occurrence of a typelist entry, add the class name to each.

```
{WT_HOME}/codebase/cfg/default/cdattachtopart.ini  
{WT_HOME}/codebase/cfg/default/associatetodocumentdialog.ini  
{WT_HOME}/codebase/cfg/default/cdbrowseitems.ini  
{WT_HOME}/codebase/cfg/default/cdaddteworkspacedialog.ini  
{WT_HOME}/codebase/cfg/default/sbdialog.ini  
{WT_HOME}/codebase/cfg/default/cdattachdocorpart.ini
```

At the following line:

```
typelist=WTPart
```

add the class name for the custom part, as follows:

```
typelist=WTPart,MYWTPart
```

2. Regenerate the wmpref.jar bootstrap file. (For more information about the bootstrap file, see the *Windchill System Administrator's Guide*.)

## Modifying Registry Files

Registry files must also be modified in order to enable custom parts. For more detailed information about registries, see the *Windchill System Administrator's Guide*.

1. Go to `${WT_HOME}/cfg/registry/objecttype`.
2. Create an object type file for the custom part. For example, you would create the following for MYWTPart:

```
wc_mywtpart.ini
```

3. Add the following entries to the object type file:

```
Author = My Company
Data Source = windchill
ID = MYWTPart
Icon = wt/clients/images/mywtpart.gif
Description = My Custom Part
Import = wc_part.ini
```

**Note:** The Data Source value must be **windchill**. The ID value must be the class name of the custom part (in this case, **MYWTPart**). The relative path name specified for the Icon value should be the value specified when the custom part was modeled (in this case, **wt/clients/images/mywtpart.gif**).

4. Copy the custom part image file, with the same relative path name as under `${WT_HOME}/codebase`, to `${WT_HOME}/codebase/cfg/default/gifs`. For example, the following file:

```
${WT_HOME}/codebase/wt/clients/images/mywtpart.gif
```

should be copied to the following location:

```
${WT_HOME}/codebase/cfg/default/gifs/wt/clients/images
```

As a result, a copy of `mywtpart.gif` now exists, with the path name `${WT_HOME}/codebase/cfg/default/gifs/wt/clients/images/mywtpart.gif`.

5. Add an entry for the new custom part object type to the [Objects] section of the following registry mode files. (For example, the entry for MYWTPART would be **Windchill MYWTPart=wc\_mywtpart**).

Workgroup Manager for CADD5:

```
${WT_HOME}/cfg/registry/mode/wmcadds5.ini  
${WT_HOME}/cfg/registry/mode/wmcadds5sb.ini
```

Workgroup Manager for CATIA:

```
${WT_HOME}/cfg/registry/mode/wmcatia.ini  
${WT_HOME}/cfg/registry/mode/wmcatiasb.ini
```

Workgroup Manager for Pro/ENGINEER:

```
${WT_HOME}/cfg/registry/mode/wmproe.ini  
${WT_HOME}/cfg/registry/mode/wmproesb.ini
```

6. Stop and restart the registry server in order to make the changes available to the Workgroup Manager client.

## Customizing Automatic Part Generation

Customizing automatic part generation is supported through the **File>Parts for Documents** menu command available within the Workspace Browser. To customize automatic part generation when using this command, however, you must create your *automatic part creator*. Before using this command, you must either create or modify your *custom automatic part creator*. This requires that you do the following:

- Create a class that supports the AutoPartGenerator interface
- Modify the AutoPartCreator preference

For more information about the **File>Parts for Documents** command, see the user's guide for your Workgroup Manager product.

### Supporting the AutoPartGenerator Interface

The AutoPartGenerator interface is part of the com.ptc.epm.commands.autopart package:

```
public interface AutoPartGenerator {  
  
    public WTPart getWTPart(EPMDocument cadDocument);  
  
    public WTPart getWTPart(EPMFamilyInstance instance);  
  
}
```

The interface methods are called during execution of the **File>Parts for Documents** command, for any selected CAD Document (EPMDocument) with no associated parts. The implementer returns an instance of WTPart, which is an existing part or new part, or null (null is a valid return for either method). If a new part is returned, it must be unique.

### Modifying the AutoPartCreator Preference

The **File>Parts for Documents** command uses the class specified as the value for the AutoPartCreator preference, which is found in the [general] section of the workspacebrowser.ini preference file.

The default setting is as follows:

```
[general]  
  
AutoPartCreator=com.ptc.epm.commands.autopart.  
    DefaultAutoPartGenerator
```

As shown in the following example, modify this value to name your automatic part creator:

```
[general]  
  
AutoPartCreator=com.ptc.epm.commands.autopart.MyAutoPartGenerator
```



# 13

## Report Generation

This chapter describes report generation tools that provide the following functionality:

- Definition of a broad range of queries against Windchill data using a graphical interface rather than Java programming.
- Use of these queries to generate reports in several output formats, including HTML, XML, and CSV.
- Customization of queries and output formats, and re-use of queries and output formats from other customizations.

To author new queries using existing report formats, you need only be familiar with the UML model for the business objects of interest for the query and have an understanding of the query definition tool. Because the report generation tools build on the capabilities of the Windchill foundation and use HTML, CSS, XML, and XSL technologies, you should also be familiar with these areas to use the tools most effectively for customization.

<b>Topic</b>	<b>Page</b>
Overview .....	13-2
Basic Report Example .....	13-2
Import and Export of Report Templates.....	13-11
Customization Details .....	13-15

## Overview

Reports are generated by applying presentation information to query result data.

Queries are built using the Windchill QueryBuilder tool and stored as a report template business object in Windchill. When a report template query is executed, it operates against the current database and produces output in the form of Java objects or XML.

Reports are generated from XML output by applying presentation transformations defined by XSLT (Extensible Stylesheet Transformation) stylesheets. The combination of XML and XSLT allows for a separation between data collection and formatting, thus facilitating separate customization of each. When defining report template objects, you can select from out-of-the-box XSLT formats or specify custom XSLT stylesheets to meet your formatting needs.

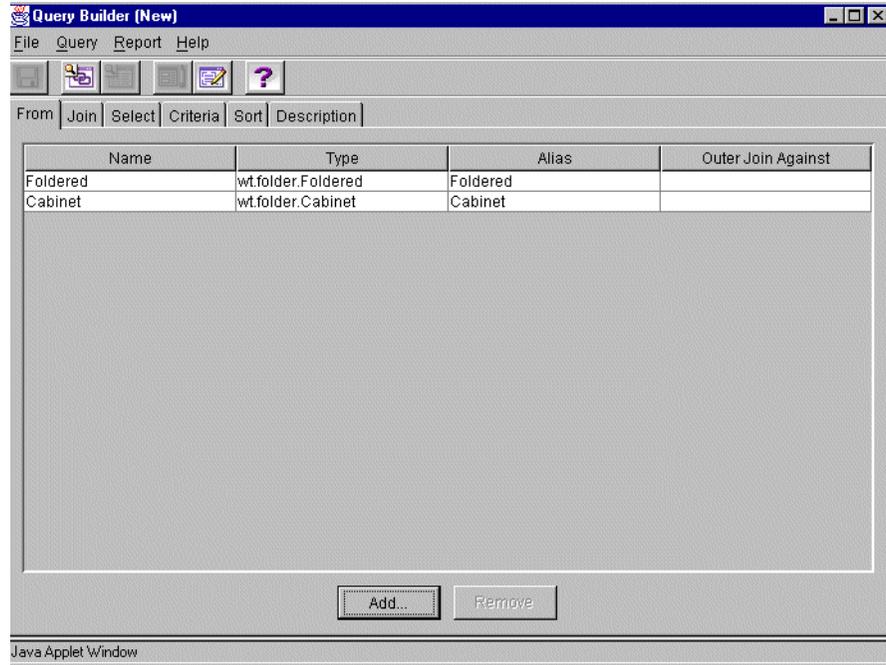
## Basic Report Example

In this example, a new report is created. The report will list all the objects in a specific cabinet and contain information similar to that displayed when browsing folders. The difference is that this report will show a flattened view as opposed to a hierarchical view. The results displayed in this example reflect demo data that has been loaded.

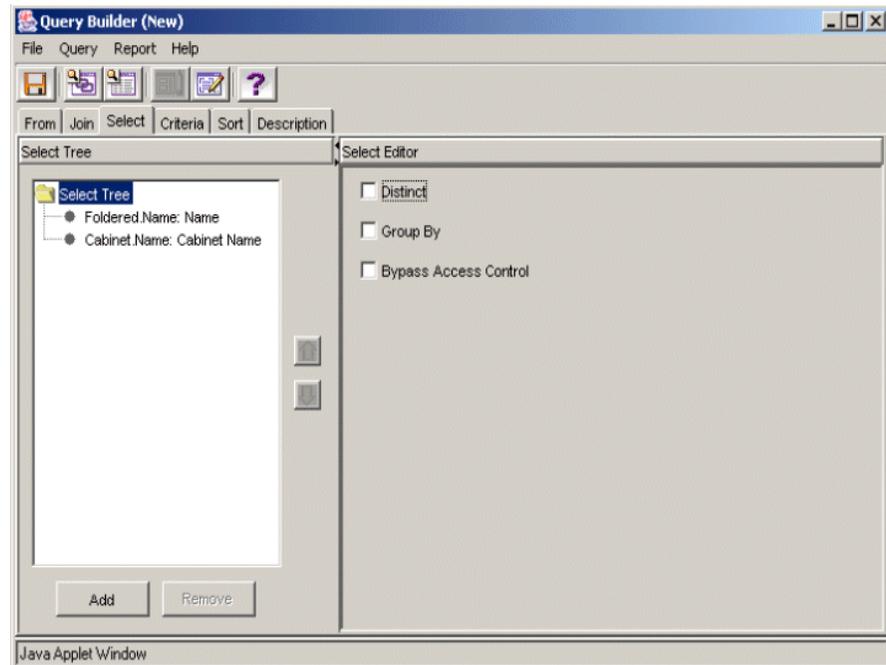
## Query

The following steps show how to create the initial query for the report:

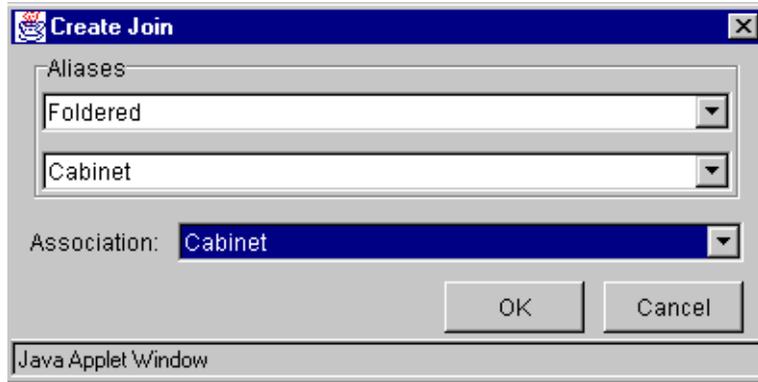
1. Create a new report template object and specify the query using the QueryBuilder user interface, as follows. (For detailed usage instructions, use the QueryBuilder online help.)
  - a. From Report Manager, click the **New** button. The QueryBuilder interface appears (see figure below).
  - b. To add classes for the query, click the **Add** button on the **From** tab. Select the **Foldered** and **Cabinet** classes. These classes will be used as the basis for the query.



- c. Select the name attribute of these classes by clicking the **Add** button on the **Select** tab.

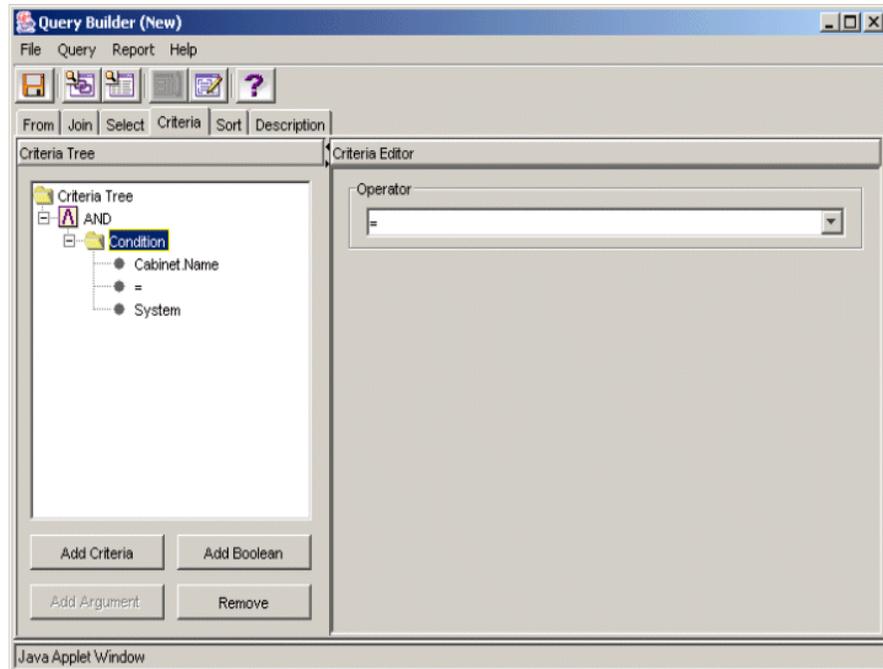


- d. Every foldered object contains a reference to its cabinet. Add a reference join between the foldered object and its associated cabinet by selecting **Query > Create Join**. In the **Create Join** dialog box, select these classes and the Cabinet reference association, then click the **OK** button.

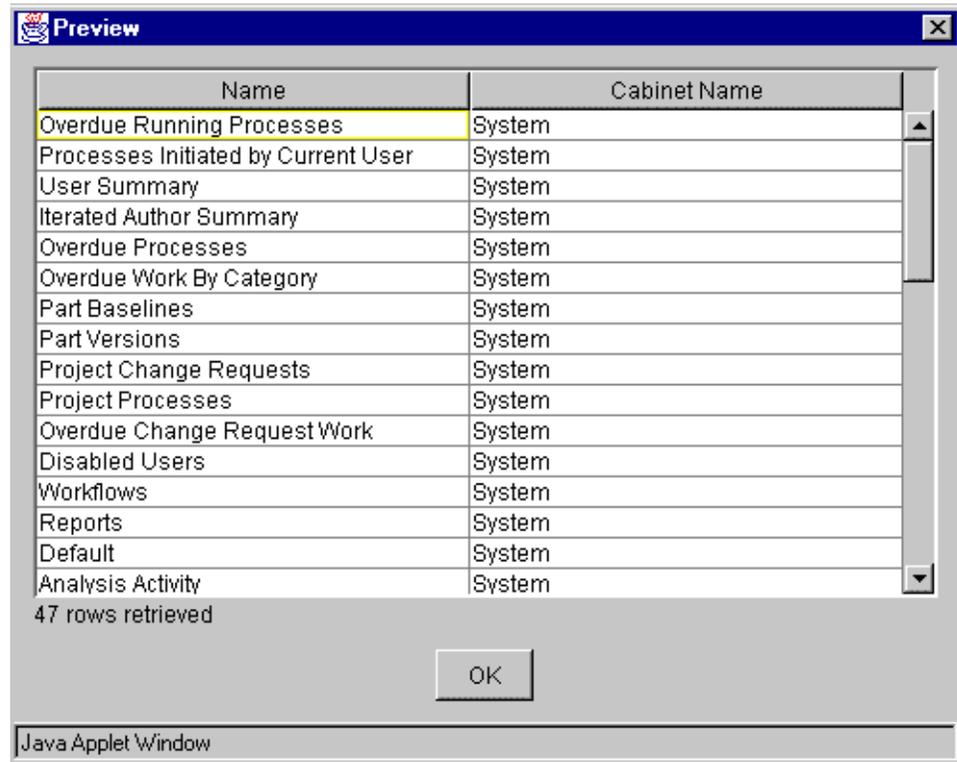


- e. Add criteria for filtering the results to only a single cabinet. On the **Criteria** tab, set the following values, as shown in the following figure:

Field	Value
Alias	Cabinet
Attribute	Name
Operator	=
Value	System

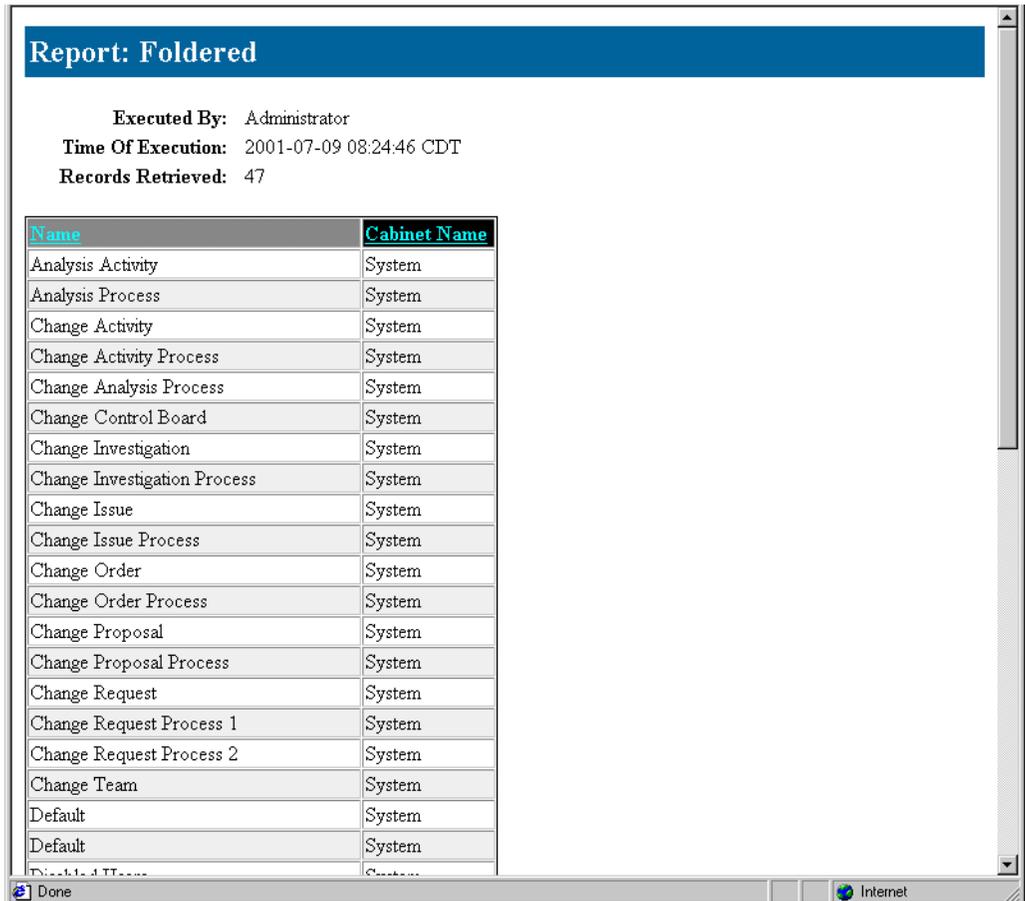


- f. Preview the data to see if the query has been constructed properly by selecting **Query > Preview**. A figure similar to the following appears.



- g. Save the report in Windchill by selecting **File > Save**.

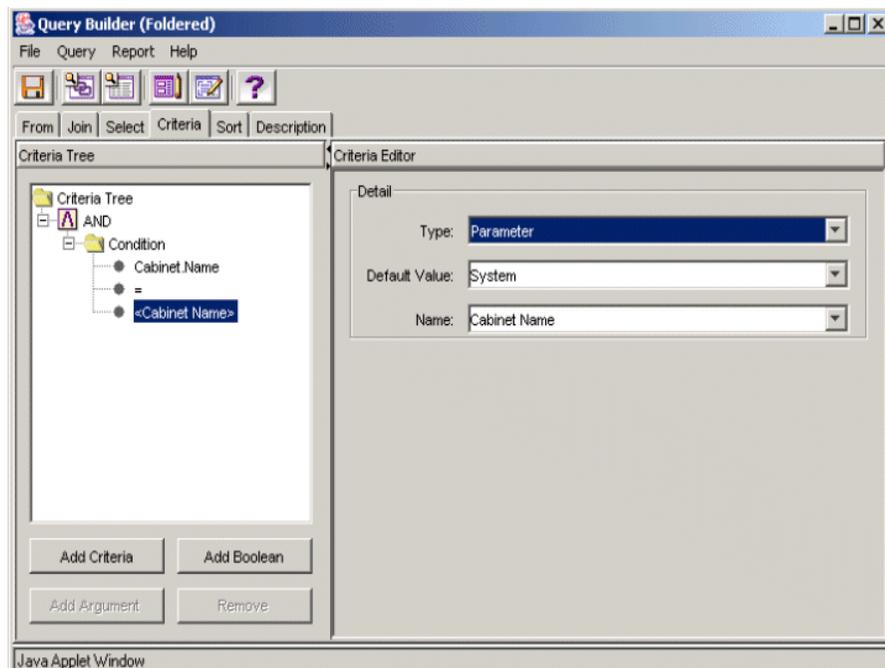
2. Execute this report as follows.
  - a. Select **Report > Generate**. (The Generate Report action is also available when viewing the properties for a report template object.)
  - b. From the report generation form, click **Standard, HTML (with sorting)**, as the output format and click the **Generate** button.



## Report Parameters

The initial report includes criteria for filtering the results based on cabinet name. In this example, the name is System. For most reports, it is desirable to have parameters that can be specified at execution time. This is accomplished using parameters in the criteria, as shown in the following steps:

1. Edit the Foldered report template to change the criteria to use a parameter, as follows:
  - a. From Report Manager, select the Foldered report and then click the **Update** button to launch QueryBuilder.
  - b. On the **Criteria** tab, change the value of the **Type** field from "Constant" to "Parameter" and specify a parameter name in the **Name** field, in this case, "Cabinet Name".



- c. Save the report template.

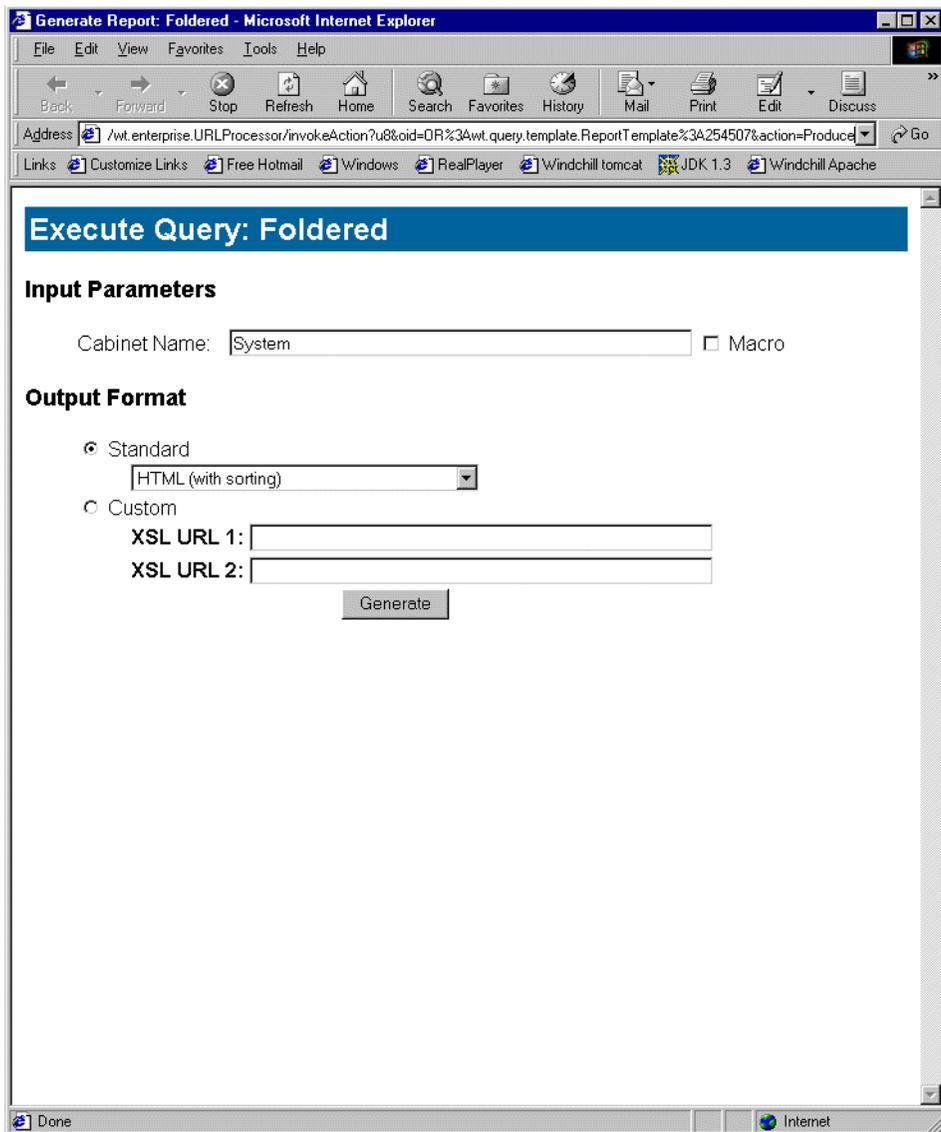
2. Execute the report, as follows:

a. Select **Report > Generate**.

The report generation form now specifies the parameter name, Cabinet Name.

b. Enter a valid cabinet name in this field, in this case, System, and click the **Generate** button.

If this field is left empty, the associated criteria is dropped. In this example, if the Cabinet Name field was left empty, the query would return all Foldered objects in any cabinet.



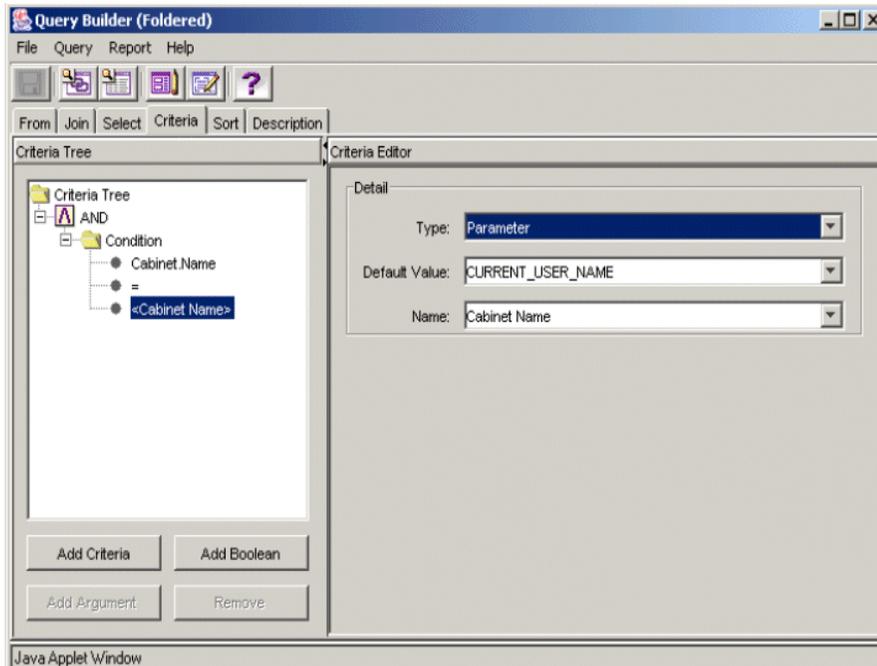
Another way to specify values dynamically is to use report parameter macros. Macros specify values that are derived by the system at report execution time. An example of a macro is `CURRENT_USER_NAME`. This macro returns the name of the current user and is useful for showing information related to the user running a report.

Macros can be specified for constants and parameters within the QueryBuilder user interface. You can also enter macro names in the report generation form. The following are predefined macros:

<b>Name</b>	<b>Description</b>	<b>Type</b>
<code>CURRENT_USER_NAME</code>	Name attribute of the current authenticated user.	String
<code>CURRENT_TIME</code>	Current system time. The time is formatted as a Java string using <code>SimpleDateFormat</code> and the method <code>server.timezone</code> (specified by <code>wt.method.timezone</code> ).	String

The following steps show how to put a macro in the report generation form:

1. Edit the Foldered report template to change the criteria to use a macro for the parameter default as follows:
  - a. Select the Foldered report and then click the **Update** button to launch QueryBuilder.
  - b. On the **Criteria** tab, change the value of the **Default Value** field from "System" to "CURRENT\_USER\_NAME" using the drop-down list.



- c. Save the report template.
2. Execute the report by selecting **Report > Generate**.

The report generation form now specifies the macro and the results show foldered items only in the current user's personal cabinet.

If demo data has been loaded, several example reports are available within the Windchill folder System/Reports. These sample reports provide examples of the types of reports that are possible. The following section contains more information on loading these sample reports.

## Import and Export of Report Templates

Two command line utilities are available for import and export of report templates to and from the database. The import utility, LoadReportTemplate, can be used in a standalone manner or from within wt.load.Demo. The export utility, ExportReportTemplate, is available in standalone mode only.

The LoadReportTemplate utility persists report template objects to the database based on input from a comma-separated value (.csv) file. For each row of the .csv file, one report template object is defined. The content of the columns in this file are defined as follows:

Column Contents	Description
ReportTemplate	The command ReportTemplate is required, as shown, in this column. It indicates to the CSV loader which load command to execute (that is, what type of data loading is to be done).
Folder	Required. This entry is the location of the report templates in Windchill Explorer. If the specified folder does not exist, one is created.
Report template name	Required.
Description	Optional.
XML source	Required. This entry specifies a path to the XML source file. This file is parsed and then validated against the QML DTD (Windchill\codebase\wt\query\qml\qml.dtd). The path is first assumed to be an absolute file path. If the file is not found, the path is then assumed to be relative to the Windchill\loadfiles directory.

Column Contents	Description
XSLT designation	<p>Possible options include the following:</p> <ul style="list-style-type: none"> <li>• DEL (delegation), which allows selection from a set of defined XSLT formats.</li> <li>• URL, which offers selection of a customized XSLT stylesheet.</li> <li>• No entry, which requires the user to select a format style at report template generation time.</li> </ul>
XSL format	<p>If the XSLT designation is DEL, this column is required. The formats currently available are as follows:</p> <p>HTML            CSV            TSV            XML            PDF            HTMLWithSorting            HTMLWithMerging            MSWord2000Portrait            MSWord2000Landscape</p>
Container path	<p>Optional. This column specifies a container where the report should be stored. If no value is specified, the site container is used by default.</p>
URL (first location)	<p>If the XSLT designation is URL, this column is required and should contain a URL pointing to a customized XSLT stylesheet.</p>
URL (second location)	<p>This column is used only if the XSLT designation is URL and, even then, is optional. It defines a second URL location for an XSLT stylesheet.</p>

An out-of-the-box .csv file, reporttemplates.csv, is provided, along with several sample XML source files. This .csv file and the XML files are used for persisting sample report templates during wt.load.Demo. The .csv file is located in Windchill\loadFiles and the sample XML files are located in Windchill\loadFiles\reports.

There are three options for using the LoadReportTemplate utility in standalone mode:

**java wt.query.template.LoadReportTemplate**

No parameters indicate that the default sample files defined earlier are being used.

**java wt.query.template.LoadReportTemplate**

***c:\LoadReports.csv***

The parameter indicates the absolute path of a user-defined .csv file.

**java wt.query.template.LoadReportTemplate**

***loadReports.csv***

The parameter indicates the relative path of a user-defined .csv file.

The ExportReportTemplate utility downloads multiple, persisted report template objects from the database to the relative file exportreports.csv. This file has the same structure as defined above.

The ExportReportTemplate utility has three optional parameters:

- The first parameter specifies a string to use to search for report template objects by name. The "%" symbol can be used as a wildcard to match any number of characters in the name. If this parameter is not specified, all report template objects are exported.
- The second parameter specifies the path of the container in which to search. If this parameter is not specified, the site container is searched to find the report template.
- The third parameter specifies whether the container should be searched hierarchically. If the value is true, the criteria will match the specified container or any of its parent containers. If the value is false, only the specified container will be matched. If this parameter is not specified, a value of true is used.

The following example exports all report template objects in the site container that have a name starting with the characters "monthly":

```
java wt.query.template.ExportReportTemplate monthly%
```

The following example exports all report template objects in the Windchill PDM, default organization, and site containers that have a name starting with the characters "monthly":

```
java wt.query.template.ExportReportTemplate monthly%  
  "/wt.inf.container.OrgContainer=  
  DefaultOrg/wt.inf.library.WTLibrary=Windchill PDM"
```

The following example exports all report template objects in the Windchill PDM container that have a name starting with the characters "monthly":

```
java wt.query.template.ExportReportTemplate monthly%  
  "/wt.inf.container.OrgContainer=  
  DefaultOrg/wt.inf.library.WTLibrary=Windchill PDM" false
```

## Customization Details

After a report exists, there are a number of ways to customize various aspects of the report. This section gives detailed information on customizing the following items:

- Query
- Report format
- Report generation client
- Report generation URL
- Macros

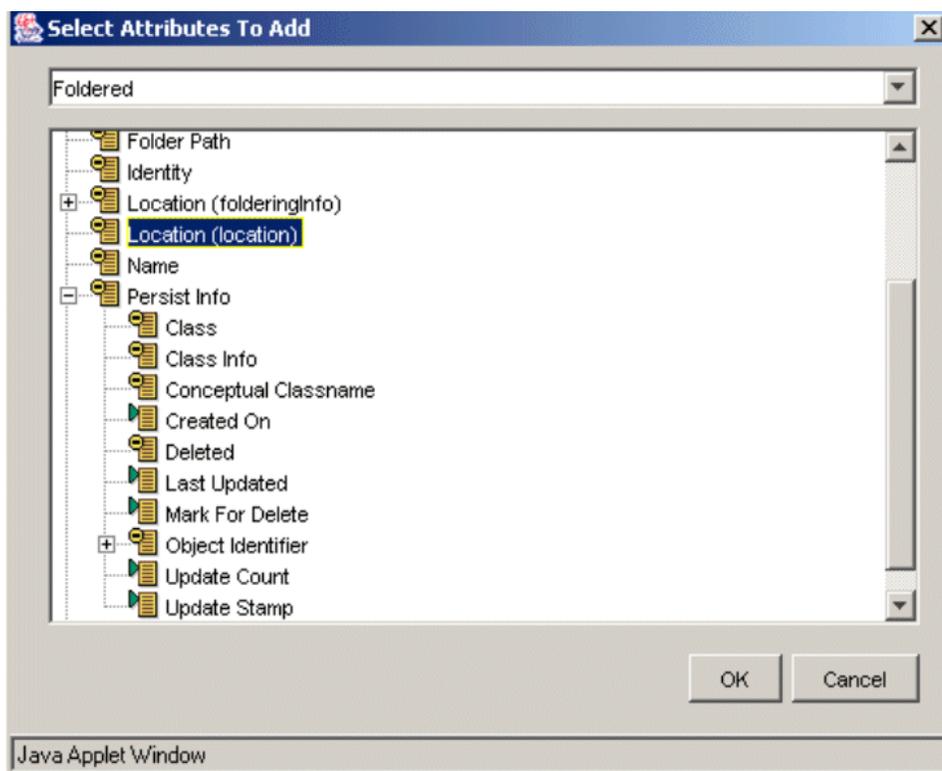
### Customizing the Query

Customizing the query involves editing the query using the QueryBuilder user interface. An existing query is opened, edited, and then saved. The new query can then be used immediately in reports.

The following steps show how to add additional attributes of the foldered object to the query results:

1. Open the existing Foldered report template object using the Report Manager **Update** button.

- From the **Select** tab, click the **Add** button. Select the Location attribute and, under Persist Info, select the Last Updated attribute. (To add multiple attributes, hold the CTRL key while selecting them.)



- Re-order the columns as necessary.
- Preview the data to make sure the query is correct.
- Save the query. The existing query is then overwritten.
- Execute the report.

## Customizing the Report Format

Several report formats are provided out-of-the-box. All of these formats are intended to be general purpose. That is, they should produce reasonable output from any query. If none of these formats are appropriate, you can modify them or create completely new formats.

## CSS Customization

All of the HTML report formats provided, except those derived from Microsoft Word, use a CSS1 stylesheet to specify font, color, size, and spacing details. Use of CSS1 allows these details to be separated from other aspects of page layout and placed into one or more re-usable stylesheets. Therefore, the easiest way to customize these aspects of the HTML reports is to edit the CSS files that they use. For further information about CSS1, refer to the W3C CSS1 specification, currently available at the following URL:<sup>1</sup>

<http://www.w3.org/TR/REC-CSS1>

Two CSS files are involved: `htmlFormat4Print.css` and `htmlFormat4Screen.css`. Both are located in `<Windchill codebase>/templates/reports`. The `htmlFormat4Print.css` file is intended to specify the appearance of HTML when printed and the `htmlFormat4Screen.css` file when viewed on screen. Actually, browsers do not support this aspect of CSS and, therefore, the printed output is controlled by `htmlFormat4Screen.css` as well. However, you may prefer `htmlFormat4Print.css` to `htmlFormat4Screen.css`, which is designed primarily for consistency with other Windchill HTML pages.

To change the CSS stylesheet used by a report or to make larger changes to the output format than possible through CSS1, you must customize XSLT stylesheets as described in the next section.

## XSLT Customization

As mentioned earlier, reports are produced by applying XSLT stylesheet transformations to XML query results. This includes all the out-of-the-box formats. XSLT stylesheets are capable of producing not only any HTML layout but also any other XML or text format.

XSLT stylesheets are based on the concepts of templates and rules. Most XSLT stylesheets are largely composed of XML or HTML tags and text that are static; that is, included verbatim in the output. The remainder of the stylesheet is then composed of rules for computing the dynamic portion of the output from the input XML. Depending on the mix of templates and rules, the nature of XSLT customization can vary from simple HTML authoring to pure programming. For the definitive specification on XSLT, refer to the W3C XSLT specification, currently available at the following URL:<sup>2</sup>

<http://www.w3.org/TR/xslt>

For additional information about XSLT, refer to the **What is XSLT?** link and the **Resources** area currently available at the following URL:<sup>3</sup>

<http://www.xml.com>

- 
1. If you have difficulty with this URL, try the site URL [www.w3.org](http://www.w3.org).
  2. If you have difficulty with this URL, try the site URL [www.w3.org](http://www.w3.org).
  3. If you have difficulty with this URL, try the site URL [www.xslinfo.com](http://www.xslinfo.com).

## Stylesheets Provided

One technique for customizing the output format is to modify one of the XSLT stylesheets that are provided. These stylesheets are described in detail in the following table. All file paths are relative to <Windchill codebase>/templates/reports.

<b>XSLT Stylesheet</b>	<b>Standard Format Name</b>	<b>Description</b>
identity.xsl	XML	Performs the identity transformation on the XML query results; that is, it outputs the XML input.
csvFormat.xsl	CSV (Comma Separated Variable)	Produces comma-separated-values format (useful for reading into spreadsheets, and so forth).
tsvFormat.xsl	TSV (Tab Separated Variable)	Produces tab-separated-values format (useful for reading into spreadsheets, and so forth).
simpleHtmlFormat.xsl	HTML	Produces a simple HTML format. Unlike the other HTML formats provided, this format does no extra formatting on numeric columns.
htmlWithSorting.xsl	HTML (with sorting)	Similar to simpleHtmlFormat.xsl except that sorting is provided through hyperlinks on the column headers.
htmlWithMerging.xsl	HTML (with merging)	Similar to simpleHtmlFormat.xsl except that vertically duplicated cells are merged under certain conditions. This is a time-consuming transformation compared to the others provided.

XSLT Stylesheet	Standard Format Name	Description
sortedHtmlAsXML.xsl		Produces the same results as htmlWithSorting.xsl except that the output is well-formed XML rather than traditional HTML, and the results contain additional non-HTML attributes. This is to facilitate application of further XSLT transforms (for example, mergeHTMLCells.xsl) to the result.
mergeHTMLCells.xsl		Merges cells in HTML (which must be well-formed XML) similar to htmlWithMerging.xsl. Additionally, the input must have extra attributes annotations. This stylesheet is not intended for use directly on the query result XML. Instead, when applied to the results of sortedHtmlAsXML.xsl, the overall effect is to produce HTML with both sorting and merging.
msw2000ls.xsl	Microsoft Word 2000 HTML (Landscape)	Produces a simple HTML format containing Microsoft Word 2000 metadata. When the result is dropped onto Microsoft Word 2000, the originally specified margins, table borders, and so on are preserved. Also, decimal tab stops are specified on floating point data columns. The result has landscape page orientation.
msw2000p.xsl	Microsoft Word 2000 HTML (Portrait)	Same as msw2000ls.xsl except the result has portrait page orientation.

XSLT Stylesheet	Standard Format Name	Description
xslfo.xsl	PDF	<p>Produces XSL Formatting Objects, an XML-based standard for high-precision page layout (see <a href="http://www.w3.org/TR/xsl">http://www.w3.org/TR/xsl</a>). This is referred to as PDF format in standard format lists because the default is to post-process this format to dynamically produce PDFs. This is done through Apache FOP (see <a href="http://xml.apache.org/fop">http://xml.apache.org/fop</a>) and is controlled by the following user preference:</p> <pre>/wt/query/report/template/post procmap/application/xslfo+xml 1</pre> <p>This format is currently limited by the capabilities of FOP (for example, column widths must be provided rather than being computed from contents) and will improve in the future as FOP improves.</p>
excel97WebQuery.xsl		<p>Produces a version of Excel Web Query (.iqy) format compatible with both Excel 97 and higher. This Excel format records the URL from which the spreadsheet data came, and can refresh the data from the URL periodically or on demand. For best results with Excel 2000, set the MIME type for Excel Web Query (IQY) files to be application/x-excel-web-query; on download, this format is then automatically loaded into Excel 2000 only. With Excel 97, you must save the downloaded file and select it from within the Excel 97 'Run Web Query...' command.</p>

XSLT Stylesheet	Standard Format Name	Description
excel2000WebQuery.xsl		<p>Produces a version of Excel Web Query (.iqy) format compatible only with Excel 2000 and higher. It uses a new feature in Excel 2000 Web queries to attempt to preserve the formatting in the spreadsheet rather than that in the HTML URL data source.</p>
barChart.xsl llineChart.xsl pieChart.xsl scatterChart.xsl		<p>Produce various types of charts corresponding to their names. The first result column is assumed to contain labels and the rest are assumed to contain chart data. Output is initially produced as SVG (Scalable Vector Graphics), a standard XML-based vector graphics format (see <a href="http://www.w3.org/TR/svg">http://www.w3.org/TR/svg</a>). Because this format requires a client viewer (for example, Adobe's free SVG Viewer available at <a href="http://www.adobe.com/svg/viewer/install/main.html">http://www.adobe.com/svg/viewer/install/main.html</a>), the default is to post-process this format to rasterize the SVG into JPEG. SVG, JPEG, or PNG output formats can be selected by the following user preferences:</p> <p>/wt/query/report/template/postprocmap/image/svg+xml</p> <p>/wt/query/report/template/postprocmap/svgRasterizer/rasterFormat</p>

XSLT Stylesheet	Standard Format Name	Description
includes/linker.xsl		Provides an XSLT template, generateURLHref, that is used by the provided HTML and XSL FO formats to produce hyperlinks to Windchill objects from XML result nodes that contain additional attribute information provided by the query layer when entire Windchill objects are selected. Object, version, e-mail, and content download links are currently supported.
includes/excelWebQueryBase.xsl		The shared implementation behind excel97WebQuery.xsl and excel2000WebQuery.xsl. This stylesheet allows stylesheets which include it to specify the relative URL of an XSLT stylesheet that produces HTML tables. The produced Excel Web Query then obtains its data by executing the report (again) but using the specified HTML table format. Thus, by changing this HTML format, one can change the data that appears in Excel as a result of this Web query. Currently both excel97WebQuery.xsl and excel2000WebQuery.xsl use simpleHtmlFormat.xsl.

XSLT Stylesheet	Standard Format Name	Description
includes/chart.xsl		The shared implementation behind barChart.xsl, lineChart.xsl, pieChart.xsl, and scatterChart.xsl. This stylesheet works by using Apache Batik (see <a href="http://xml.apache.org/batik">http://xml.apache.org/batik</a> ) on the server to capture JChart graphics as SVG and optionally rasterize them. Note that custom charts and graphics can easily be captured as well by implementing <code>wt.query.template.Chartable</code> interface and replacing "wt.query.template.ChartCreator" in this file with the name of the custom Chartable implementation.
includes/msw2000.xsl		The shared implementation behind msw2000ls.xsl and msw2000p.xsl. The page size, margins, and orientation are all specified as inputs. This stylesheet is intended solely for inclusion from other stylesheets.
includes/htmlWithSortingBase.xsl		The shared implementation behind htmlWithSorting.xsl and sortedHtmlAsXML.xsl. This stylesheet is intended solely for inclusion from other stylesheets.
includes/localize String.xsl		Provides an XSLT named template (that is, a macro) for localizing strings. The implementation uses an XSLT extension function to call back into Java to access Java resource bundles. This stylesheet is intended solely for inclusion from other stylesheets.

XSLT Stylesheet	Standard Format Name	Description
includes/urlEncode.xsl		Provides an XSLT named template for URL encoding. The implementation uses an XSLT extension function to call WTURLEncoder.encode in Java. This stylesheet is intended solely for inclusion from other stylesheets.

Due to the template-based nature of XSLT, some modifications can be made to the provided stylesheets based almost solely on a knowledge of the intended output format (such as HTML). For example, the provided formats that use CSS1 stylesheets generate the references to these files using the following lines in the XSLT stylesheets:

```
<link rel="stylesheet" type="text/css"
      href="{ $windchill }/templates/reports/htmlFormat4Screen.css"
      media="screen" />

<link rel="stylesheet" type="text/css"
      href="{ $windchill }/templates/reports/htmlFormat4Print.css"
      media="print" />
```

The only part of these lines that is not strictly HTML is the portion within the { } braces which, though similar to JavaScript expression usage, is actually an XSLT XPath expression. In this case, the expression is referencing the variable windchill, which was previously assigned the URL of the Windchill codebase.

## XML Resource Bundles

One aspect of the provided XSLT stylesheets (which is shared with those for the Windchill Product Configurator) is that they use XML resource bundles for localization of text. Rather than producing different versions of each XSLT stylesheet for each locale, the locale-dependent text has been extracted into separate XML files. These are loosely referred to as XML resource bundles, and are retrieved and searched through XSLT based on the locale.

This approach was taken for the following reasons:

- To allow ease of localization by using the same technology, syntax, and tools as those behind XSLT and HTML authoring; that is, XML.
- To allow a tighter coupling between XSLT stylesheets and their localization data. Because relative URLs are used to look up XML resource bundles, they can be installed relative to the XSLT files that reference them, rather than relative to the Windchill codebase.
- To allow localization without dependence on XSLT extension functions.

The XML resource bundles used by the provided XSLT stylesheets are located at `<Windchill codebase>/templates/reports/defReportRB_*.xml`.

If you decide to use this technique in your own XSLT stylesheets, see the XML resource bundles provided and the XSLT files provided for usage examples. Keep in mind the following issues:

- XML resource bundles must be well-formed XML. For further information, refer to XML.com's annotated XML specification, currently available at the following URL:<sup>4</sup>

<http://www.xml.com/pub/axml/axmlintro.html>

The W3C unannotated, original version of the XML specification is currently available at the following URL:<sup>5</sup>

<http://www.w3.org/TR/REC-xml>

- The encoding listed in the `<?xml ...?>` header should match that actually used when saving the file and should be one supported by the XML parser used (Xerxes 1.2.2). For example, on Windows NT, you can use the **Save as Unicode** option in NotePad and specify an encoding of "unicode" in the XML resource bundle.

---

4. If you have difficulty with this URL, try the site URL [www.xml.com](http://www.xml.com).

5. If you have difficulty with this URL, try the site URL [www.w3.org](http://www.w3.org).

## New Formats

Besides modifying existing XSLT, you can also author entirely new XSLT stylesheets. The XSLT standard is still relatively new, however, and therefore editors that accurately display the actual appearance are not yet widely available. You can, however, author new stylesheets without any such tools using the following steps:

1. Author a sample of the intended output format. For example, use an HTML editor to produce a sample of the intended format; create a sample document in Microsoft Office and save it as HTML (this is how the Microsoft Word-based HTML formats were created); or export an Adobe Illustrator drawing as SVG.<sup>6</sup>
2. Copy static pieces of the sample output to a skeleton XSLT stylesheet, editing or escaping them as necessary to ensure that they are well-formed XML. (All XSLT must be well-formed XML.)
3. Author XSLT to transform the input XML into the dynamic portions of the output, using the sample output as a guide.

The number and complexity of transformations to be done in step 3 can vary greatly from one format to another and largely determine the effort involved in creating a new format. XSLT provides extensive capabilities for sorting, filtering, summing, and combining XML data. Additionally, you can provide XSLT extension functions and elements that call other languages, including Java and JavaScript. See the Saxon documentation for further details.

---

6. The use of Apache Batik, as described in the various chart.xml stylesheets (see the table in the section on [Stylesheets Provided](#) earlier in this chapter), provides an alternative means of producing SVG with minimal knowledge of XSLT.

## XML Result Format

Understanding the query result XML format is extremely important to successfully retrieving the desired data from it using XSLT (or any other technique). An outline of this format is shown below. The portions in bold represent dynamic data and ellipses (...) represent omitted items.

```
<?xml version="1.0" encoding="UTF-8" ?>
<queryResult>
  <metadata>
    <name>Report Name</name>
    <description>Report Description</description>
    <objectIdentifier>Report Object Id</objectIdentifier>
    <sourceSystem>Windchill Codebase URL</sourceSystem>
    <sourceGateway>Windchill Gateway URL</sourceGateway>
    <executingPrincipal>
      <fullName>User's Full Name</fullName>
      <objectIdentifier>User Object ID</objectIdentifier>
    </executingPrincipal>
    <timeOfExecution>Time when Executed</timeOfExecution>
    <locale>Locale of User</locale>
  </metadata>
  <auxData>
    <dataItem name="requestParam1">value</dataItem>
    . . .
    <dataItem name="requestParamN">value</dataItem>
  </auxData>
  <qml>
    . . .
  </qml>
  <actualParameter parameterId="ParamName1"> Value
  </actualParameter>
  . . .
  <actualParameter parameterId="ParamNameN"> Value
  </actualParameter>
  <result>
    <heading tag="column1TagName"
      type="Java data type">column header</heading>
    . . .
    <heading tag="columnNTagName"
      type="Java data type">column header</heading>
    <row idx="1">
      <column1TagName>value</column1TagName>
      . . .
      <columnNTagName>value</ColumnTagName>
    </row>
    . . .
    <row idx="N">
      . . .
    </row>
  </result>
</queryResult>
```

If an entire Windchill top-level object (a Persistable) is selected, additional attributes are generated on the column tag element, including the object ID, branch ID (if the object is versioned), and classname of the object. WTUser objects also include email attributes. These attributes are generated primarily to facilitate generation of hyperlinks. Out-of-the-box XSLT stylesheets for HTML and PDF formats use these attributes whenever they are present to produce hyperlinks to the objects.

Additional flexibility for generating hyperlinks is provided by use of the characters \$\$ in column names. If a column name containing \$\$ is specified through the QueryBuilder, an individual cell is not created for it; instead, the name is parsed as follows. If the part of the column name preceding \$\$ matches another column name that does not contain \$\$, its data is added to the other column as an attribute, which is determined by the part following \$\$.

For example, assume the following columns are specified through QueryBuilder:

- Part
- Part\$\$branchID
- Part\$\$type

Each XML row would then contain a Part element like the following:

```
<Part branchID="dataFromBranchIdColumn"
      type="dataFromTypeColumn">dataFromPartColumn</Part>
```

Rows would not contain individual Part\$\$branchID or Part\$\$type columns. This functionality allows you to select the right data to obtain hyperlinks (as described earlier in this section) without having to select entire Persistable objects.

If the column type is java.util.Date, column data is formatted based on the HTTP request's Locale. In addition, the column element has a "value" attribute containing the raw Java string value. If other date-formatting customization is needed, this value could be used.

The table below summarizes the data conveyed by the various top-level elements in the query result XML.

Element	Description
metadata	Meta-information about the query from which the data resulted and the user executing it.
auxData	The request parameters used.
qml	Fully defines the details of the report template query which was made (see <Windchill codebase>/wt/query/qml/qml.dtd for further details).
actualParameters	The query parameters used when executing the query.
result	The column headers and types, and the rows of data.

## Using XSLT Stylesheets as Report Formats

To use an XSLT stylesheet as a report format, you must specify it either in the report template user interface or, for report templates that specify their format as **Ask Upon Generate**, upon generation using the HTML report generation form. In either case, two means of specifying formats are allowed: custom and standard.

Specification of custom formats is done using up to two relative or absolute URLs. When the URLs are not absolute, they are relative to the Windchill codebase. When two XSLT URLs are specified, they are applied to the report results in series. The custom specification mechanism allows use of XSLT stylesheets that are not located on the Windchill server and facilitates testing of new stylesheets. It also allows the application of two XSLT stylesheets in a series, which is not possible through the standard format mechanism.

Specification of standard formats is done by selecting one from a list. Most of the provided XSLT stylesheets are part of this list. Additional formats can be added to this list by editing `dbservice.properties`. These property entries begin with `wt.services/rsc/default/wt.query.report.DefaultXSL`. The format of these entries is that of the Windchill application services delegation mechanism. For further information, see Chapter 5, Customizing `service.properties`. If localization of the additional entries is required, edit the Java resource bundle `wt.query.template.TemplateResource`.

## Customizing the Report Generation Client

The report generation client consists of an HTML form that can prompt the user for additional report generation input and an HTTP processor that executes the report and applies the XSL transformations. Both of these clients can be customized in the same manner as typical HTML client customizations. (For further information, see Chapter 7, Customizing the HTML Client.)

In addition, report generation-specific code is available as static methods in a separate class so it can be reused. The `ReportTemplateHelper` class provides many methods for processing reports (see the Javadoc for `wt.query.template.ReportTemplateHelper`). Reports that use soft types (that is, types created using the Type Manager) should be processed using the report commands (see the Javadoc for the `com.ptc.core.query.report.command.common` package).

Both of the clients mentioned above rely on a wrapper API developed for XSLT processors. The API can be found in the `wt.xml.xslt` package, and is documented in the Windchill UML model and Windchill API Javadoc.

This API provides the following functionality:

- Independence from individual XSLT implementations (for example, Saxon, the XSLT processor library currently in use)
- A high-level abstraction for XML source that hides the details of a particular implementation (for example, String, Java IO stream, DOM, or SAX)

- A clean API for XSLT operations in Windchill
- Easy-to-use, high-level facilities for complex chaining of XSLT transformations

For additional capabilities beyond those provided through this API, you can use the standard JAXP (Java API for XML Processing) APIs, which are part of Java 2 v1.4, or access Saxon directly. For further information, see the Saxon Web page, currently available at the following URL:

<http://saxon.sourceforge.net/>

Note, however, that the XSLT library bundled with Windchill may change in the future and that users of the Windchill XSLT and JAXP APIs will be affected less by any such change.

## Report Generation Form

The report generation form is used to gather additional report input parameters prior to execution. It consists of two main sections: report parameters and output format.

The report parameters section is built from data specified in the query. Given a report query as an XML source, the `ReportTemplateHelper buildParameterTemplates( )` method returns an array of `ParameterTemplate` instances that represent the parameters of the query. The current client accesses this parameter template data to build the form to prompt the user for input. Note that the processing bypasses any parameter templates if the specified name is included in the HTTP parameters (see the section Customizing the Report Generation URL that follows). This is an indirect way to customize the form. For example, fully specifying all parameter values in the report generation URL would cause no parameter input fields to be generated. Also, the current form processing completely bypasses the form and goes directly to the report execution processing if the XSL specification has been set in the report template and all required parameters (if any) have had values specified in the HTTP parameters.

The output format section is generated based on the XSL specification of the report template object. If one exists, the output format section is bypassed. Otherwise, the input fields for specifying the format are generated. The `ReportTemplateHelper getAvailableXSLFormats( )` method is used to build the drop-down choice for the standard XSL.

## Report Generation Output

The report generation output relies on the XSLT API. The basic steps are as follows:

1. Generate report data as XML using the ReportTemplateHelper generateXML() method.
2. Obtain a list of XSL stylesheets to apply. This can be specified as an attribute of the report template object or as HTTP parameters. The report template object XSL Specification attribute has precedence over the HTTP parameter values. This logic is implemented in the ReportTemplateHelper.getXSLSpec() method.
3. Create XSLT Transform objects chaining together the array of Stylesheets as necessary. This method is implemented in ReportTemplateHelper.getTransform().
4. Get the post-processor based on the output media type using ReportTemplateHelper.getPostProcessor(). This method uses the preferences mechanism to retrieve the class name of the post-processor.
5. Set the response output type and generate data to the response output stream:

```
if(postProcessor==null)
{
    // No post processor found
    response.setHeader("content-type",
        ReportTemplateHelper.concatMediaTypeAndEncoding(
            outputMimeType, finalSheet.getOutputEncoding());
    transform.outputToStream( response.getOutputStream() );
}
else
{
    // Post processor found
    response.setHeader("content-type",
        ReportTemplateHelper.concatMediaTypeAndEncoding(
            postProcessor.getOutputMediaType(),
            postProcessor.getOutputEncoding());
    postProcessor.process( transform, response.getOutputStream() );
}
```

## Creating a New Client

The preceding sections describe the current report generation client and some simple customization points. Using the XSLT API, ReportTemplateHelper methods, and report commands as a basis, there are many ways that an entirely new client could be created. However, detailed discussions on this topic are beyond the scope of this document.

The following are possible example customizations:

- Use the ReportTemplateHelper APIs or report commands that return TypeInstances, QueryResult or Java Swing TableModel objects. These methods can be used in a Java client that uses a specific type of display or must implement additional result set processing.
- Execute report generation in a background or batch mode (for example, running reports nightly) and saving the results in Windchill.

## Customizing the Report Generation URL

Understanding the report generation URL is important for customizing applications to seamlessly integrate report generation functionality. The URLs that invoke reports can be created dynamically or set up statically (for example, bookmarked) with known URL parameters. There are two types of report generation URLs: Generate Form and Execute Report. The Generate Form URL requests additional input from the user (only if necessary) and generates a form action button that leads to the Execute Report page. The Execute Report page executes the query and returns output to the HTTP request output stream.

Both of these URLs can be edited directly to add URL parameters. Either the original or the edited URL can be bookmarked or used in a hyperlink for future reuse. The URLs can also be generated programatically. The ReportTempateHelper class provides APIs for each type of URL. Several versions, each taking different parameters, are also available. For further information, see the Javadoc for `getGenerateFormURL()` and `getExecuteReportURL()`.

The Generate Form URL can have parameter values specified directly. For any parameter names that are specified on the URL, the input field is bypassed. In addition, all URL parameters are passed to the Execute Report URL through the form's action button. The Execute Report URL also derives all report parameter values from the URL parameters (through ReportTemplateHelper methods). If a URL parameter does not exist for a report parameter, then the report parameter's default value is used. In addition, several other URL parameters, described in the following table, can be used.

URL Parameter Name	Description
jrb	Name of a Java resource bundle for localization of result column headers. The provided XSLT stylesheets are all capable of localizing the result column headers if the name of a Java resource bundle is provided to them. This is intended to allow for re-use of a single query to support multiple locales because the report template user interface provides no means of entering strings per locale.
format	Specifies the format type: formatDelegate or formatCustom. This parameter is used only if the report template object has no XSL specification.
delegateName	Specifies the delegate name to obtain an XSL stylesheet from Windchill codebase. The delegate name must match a value specified in the dbservice.properties file. This parameter is used only if the report template object has no XSL specification and the <b>format</b> URL parameter has a value of "formatDelegate".
xsl1	Specifies a URL for a custom XSL stylesheet. This stylesheet is applied immediately to the query result and can generate direct output to the user, or output suitable for input to another XSL stylesheet. This parameter is used only if the report template object has no XSL specification and the <b>format</b> URL parameter has a value of "formatCustom".

URL Parameter Name	Description
xsl2	Specifies a URL for a custom XSL stylesheet. This stylesheet is applied to the output of the first XSL stylesheet (specified by xsl1) and should generate output to the user. This parameter is used only if the report template object has no XSL specification, the <b>format</b> URL parameter has a value of "formatCustom", and the xsl1 URL parameter has been specified.

## Customizing Macros

Customizing macros uses the standard Windchill application services delegation mechanism (for further information, see Chapter 5, Customizing service.properties). This customization example creates a new macro to automatically compute a cutoff time. A new query is created that uses this macro and generates a report containing objects that have been modified within the last three days.

Note that this example assumes you are familiar with Rational Rose and Windchill code generation, and have completed the previous customization examples.

1. Create a new implementation of the MacroExpressionProcessor interface using Rational Rose and Windchill code generation by performing the following steps:
  - a. Create a new package or use an existing one. Name the package, for example, myPackage.
  - b. Create a new class and inherit from the MacroExpressionProcessor interface in the wt.query.report package. Name the class, for example, TimeCutoffMacroProcessor.
  - c. Generate the code for this class.

- d. Fill in the implementation of the buildExpression( ) method. This implementation reads the current system time in milliseconds, computes the new time, and creates a new Date. The Date value is returned as a wt.query.DateExpression. This is necessary because of special handling of dates in SQL expressions to take into account Java representation of dates and timezone settings.

```
final int DAYS = 3;

long currentSeconds = (System.currentTimeMillis() / 1000);
long timeSeconds = currentSeconds - (60 * 60 * 24 * DAYS);

java.util.Date time = new java.util.Date(timeSeconds * 1000);
return new DateExpression(time);
```

- e. Fill in the implementation of the getValue( ) method. This value is the actual Date value as computed in the preceding step.
2. Create a logical name for the macro and map it to the implementation class. For example, for logical name "TIME\_CUTOFF" and class "myPackage.TimeCutoffMacroExpressionProcessor", the entry would be as follows:

```
wt.services/svc/default/wt.query.report.MacroExpressionProcessor/
TIME_CUTOFF/java.lang.Object/
0=myPackage.TimeCutoffMacroProcessor/singleton
```

3. Create a new report query that uses the new macro by performing the following steps:
  - a. Open the existing Foldered query and save it as a new query, for example, FolderedModified.
  - b. Remove the criteria based on cabinet name.
  - c. Add criteria. On the **Criteria** tab, set the following values as shown in the following table:

Field	Value
Class	Foldered
Attribute	thePersistInfo.modifyStamp
Operator	>
Value	TIME CUTOFF

Note that the time cutoff macro now appears in the drop-down list.

- d. Save the query.
4. Execute the report.

## Customizing QueryBuilder Types

QueryBuilder uses model information to present the list of all possible types to build queries against. It does this by searching for all types that implement the `wt.fc.NetFactor` interface. However, there may be modeled interfaces that do not implement this interface, yet have subclasses that implement a `Persistable` interface and can be used in queries. An example in the Windchill model is `wt.index.Indexable`. To allow queries to be built using these types, these types can be listed as resources in a service properties file (for further information, see Chapter 5, Customizing `service.properties`). The out-of-the-box entry in `dbservice.properties` handles the `Indexable` interface as follows:

```
wt.services/rsc/default/wt.query.report.ClassName/  
wt.index.Indexable/java.lang.Object/0=wt.index.Indexable
```

To customize the list of QueryBuilder types, new entries can be added. For example, for the class `myPackage.myClass`, the entry would be as follows:

```
wt.services/rsc/default/wt.query.report.ClassName/  
myPackage.myClass/java.lang.Object/0= myPackage.myClass
```

This is necessary only if the class does not implement the `NetFactor` interface, but does have subclasses that implement the `Persistable` interface.



# 14

## Customization Examples

This chapter is intended to contain examples of typical customizations. Currently, it gives an example of how to create a listener service. The example assumes you are familiar with Rational Rose, Visual Café, and the general concepts described in the *Windchill Application Developer's Guide*.

Additional customization examples are also given in chapter 7, Customizing the HTML Client; chapter 8, Customizing Change Management; and chapter 9, Customizing Foundation Applications.

<b>Topic</b>	<b>Page</b>
Creating a Listener Service .....	14-2

## Creating a Listener Service

A listener is a type of Windchill service. It has no methods that can be called by clients, but instead registers a listener object that listens for events and responds to them.

This example creates the projectaux (Project Auxiliary) package, which implements the projectaux service. The projectaux service listens for an event that signifies the creation of a project, and then creates a folder for that project in the cabinet containing the project object.

This section describes how to create the service, test it, and add additional events.

### Creating the projectaux Service

Creating the projectaux service consists of four major steps:

- Modeling the projectaux package and generating code
- Modifying the generated Java code
- Compiling the Java code
- Registering the service

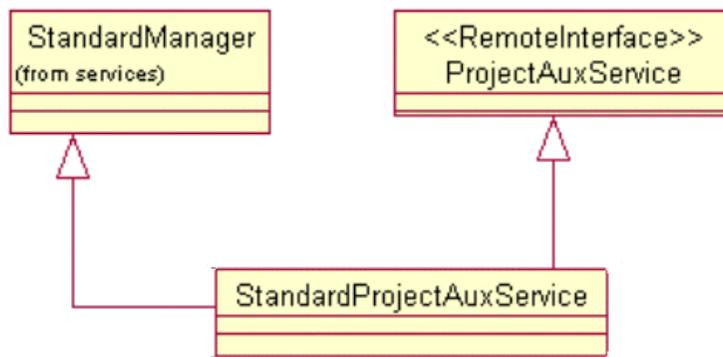
The code for this example is available in the `$WT_HOME\src\customization\projectaux` directory.

### Modeling the projectaux Package

To create the projectaux package, perform the following steps:

1. In the Windchill Information Modeler, create a package for the new service (the example code provided uses the `customization.projectaux` package).

Draw the model as shown in the following figure:



2. Create the two new classes (ProjectAuxService and StandardProjectAuxService) and the two generalization associations. You need name only the classes; there are no new attributes or operations to add.
3. Generate the code.

## Modifying the Generated Java Code

Make the following modifications to StandardProjectAuxService.java. Code to be added is indicated by bold .

### Import Statements

```
///begin user.imports preserve=yes
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

import java.util.MissingResourceException;
import java.util.ResourceBundle;
import java.text.MessageFormat;

import wt.events.KeyedEvent;
import wt.events.KeyedEventBranch;
import wt.events.KeyedEventListener;

import wt.fc.PersistenceHelper;
import wt.fc.Persistable;
import wt.fc.ObjectReference;
import wt.fc.PersistenceServerHelper;
import wt.fc.PersistenceManagerEvent;

import wt.folder.FolderHelper;

import wt.project.Project;

import wt.services.ManagerService;
import wt.services.ManagerServiceFactory;
import wt.services.ServiceEventListenerAdapter;
import wt.services.ManagerException;

import wt.util.WTProperties;
import wt.util.WTContext;
import wt.util.WTPropertyVetoException;

///end user.imports
```

### Listener Private Instance Variable

```
///begin static.initialization preserve=yes
private KeyedEventListener listener;
///end static.initialization
```

## User Operations

Add the following classes and methods:

- An internal listener class, `ProjectAuxEventListener`, which is a subclass of `wt.services.ServiceEventListenerAdapter`.
- A `performStartupProcess()` method, which overrides the method provided by `wt.services.StandardManager`. This method is called automatically when the service is registered with the method server. It creates an instance of the listener class and registers it.
- A `ProcessPostStoreEvent()` method, which responds to a particular event, in this case, the completion of the storage (persistence) of a new project object.

The code for all three items follows and is also available in the `src\customization\projectaux` directory:

```
//#begin user.operations preserve=yes

/*
 * Internal Listener class
 */
class ProjectAuxEventListener extends ServiceEventListenerAdapter {

    public ProjectAuxEventListener (String manager_name) {
        super (manager_name);
    }

    public void notifyVetoableEvent (Object event)
        throws WTEException {
        if (!(event instanceof KeyedEvent)) {
            return;
        }
        KeyedEvent keyedEvent = (KeyedEvent) event;
        Object target = keyedEvent.getEventTarget ();

        if ( keyedEvent.getEventType( ).equals(
            PersistenceManagerEvent.POST_STORE ) ) {
            processPostStoreEvent( ( Persistable )target );
        }
    }
}

protected void performStartupProcess( ) throws ManagerException {
    listener = new ProjectAuxEventListener(
        this.getConceptualClassname ( ) );

    getManagerService( ).addEventListener( listener,
        PersistenceManagerEvent.generateEventKey(
            PersistenceManagerEvent.POST_STORE ) );
}

protected void processPostStoreEvent( Persistable target )
    throws WTEException {
    if ( target instanceof Project ) {
```

```

    Project project = ( Project ) target;
    String projectName = project.getName();
    String projectLocation = project.getLocation();
    String newSubFolderName = projectLocation + "/" + projectName;

    FolderHelper.service.createSubFolder(newSubFolderName);
}
}

//##end user.operations

```

## Compiling the Java Code

Compile the following three Java files:

- ProjectAuxService.java
- ProjectAuxServiceFwd.java
- StandardProjectAuxService.java

You need modify only StandardProjectAuxService.java as described in the preceding section.

## Registering the Service

Finally, you must register your service with the method server. This is done by adding an entry to your wt.properties file. Following is an example, where the new entry is indicated by bold type:

```

wt.services.service.46 =
    wt.csm.navigation.service.CsmIBAHolderListenerService/
    wt.csm.navigation.service.StandardCsmIBAHolderListenerService
wt.services.service.47 =
    wt.units.service.UnitsService/
    wt.units.service.StandardUnitsService
wt.services.service.48 =
    customization.projectaux.ProjectAuxService/
    customization.projectaux.StandardProjectAuxService

```

## Testing the projectaux Service

To test the new service, you must restart your method server to ensure that the new service is registered. Once that is completed, create a new project using the Windchill Project Administrator (the Windchill user you log in as must have permission to create projects).

In the Create Project dialog, you must specify a location for the project. It is in that location that your new service (projectaux) will create a new folder with the same name as the project. After you have created the project, use the Windchill Explorer to verify that the new folder was created in the specified location.

## Adding Events to the projectaux Service

Windchill services emit many events. Following are a few ways to determine what events are emitted. To learn more about a particular event, see the Javadoc for the associated class.

### KeyedEvent Class

Most events are represented as public constants on subclasses of `wt.events.KeyedEvent`. For an example, see the Javadoc for `wt.fc.PersistenceManagerEvent`.

### Event Tree

Events are stored in Windchill in an event tree or hierarchy. A complete list of registered events can be generated programmatically by printing this event tree. To do so, call the following line of code from `StandardProjectAuxService` (for example, you could temporarily insert it into the `processPostStoreEvent()` method shown earlier):

```
getManagerService().printAllEventBranches()
```

To add more events to your listener service, perform the following steps:

1. In the `performStartupProcess()` method, register the event listener by calling the `addEventListener()` method again. You can use the same listener instance as before, just specify a different event. For example, you might want to listen for two events, as follows:

```
getManagerService().addEventListener( listener ,
    PersistenceManagerEvent.generateEventKey(
        PersistenceManagerEvent.POST_STORE ) );

getManagerService().addEventListener( listener ,
    PersistenceManagerEvent.generateEventKey(
        PersistenceManagerEvent.POST_DELETE) );
```

2. In the `notifyVetoableEvent()` method of your listener, create an "if" statement to check for and respond to the new event type.
3. Create a method which will be called from the "if" statement to respond to the new event type, similar to the `processPostStoreEvent()` method shown earlier in this section.

# 15

## Xconfmanager Utility

This chapter contains information about the xconfmanager utility.

<b>Topic</b>	<b>Page</b>
About the xconfmanager Utility.....	15-2
About the Windchill Command .....	15-5

## About the xconfmanager Utility

The xconfmanager is a command line utility that you can run to add, remove, or modify properties in any Windchill property file. With one exception, the following files are managed by Windchill Information Modeler and should not be edited manually or edited with the xconfmanager:

- associationRegistry.properties
- classRegistry.properties
- descendentRegistry.properties
- modelRegistry.properties

The xconfmanager utility saves your changes in the site.xconf file and provides an option to generate updated property files using the updates in the site.xconf file. The site.xconf file contains the changes made to Windchill property files starting with the installation and continuing with each use of the xconfmanager utility or the System Configurator. The xconfmanager utility is located in the *<Windchill>/bin* directory.

This chapter describes only the information and instructions necessary to modify specific Windchill properties. A full description of the xconfmanager utility and management of the Windchill property files is documented in the *Windchill System Administrator's Guide* in the Administering Runtime Services chapter.

Anyone with write access to the XCONF and property files under the Windchill installation directory can successfully run the xconfmanager utility. The xconfmanager is executed from the command line from within a windchill shell. See About the windchill Command chapter for more information about the windchill shell.

The syntax of xconfmanager command is as follows:

```
xconfmanager {-FhuwvV} {-r <product_root>} {-s <property_pair>
{-t <property_file>}} {--reset <property_names>}
{--undefine <property_names>} {-d <property_names>} {-p}
```

For the purposes of modifying Windchill properties, you will primarily use the set (s), targetFile (t), and propagate (p) parameters.

- set is used to define the property and new property value. See the Formatting Property Value Guidelines section (below) for information about formatting the *<property\_pair>* value.
- targetFile is used to specify the directory location of the property file. If the file name or path contains spaces, you must enclose the *<property\_file>* value in double quotes (" "). It is recommended to use a fully-qualified file name to ensure an accurate reference to the file is made.
- propagate is used to propagate the changes made to the XCONF files into the property file being modified in order to keep the XCONF and the property files in synch with one another.

- `help` is used to view the help for `xconfmanager`.

Some examples are as follows:

- `xconfmanager` is run from the windchill shell. To open a windchill shell, at a command prompt, execute the following command:

```
windchill shell
```

- To display `xconfmanager` help, execute the following command from the windchill shell:

```
xconfmanager -h
```

- To display the current settings for a property, execute the following command from the windchill shell:

```
xconfmanager -d <property_names>
```

- To change a property value, execute the following command from the windchill shell:

```
xconfmanager -s <property_pair>=<property_value>  
-t <property_file> -p
```

It is recommended to use the fully-qualified name of the property file to ensure an accurate reference to the file is made.

## Formatting Property Value Guidelines

The property values you set must conform with the specification for `java.util.Properties`. The following guidelines will help ensure that you set properties correctly:

Use forward slashes (/) in file paths so that the platform designation is not an issue.

To specify a property whose value contains characters that might be interpreted by your shell, escape them using the appropriate technique for the shell you are using.

For example, on a Windows system you can include spaces in a value by enclosing the argument with double quotes. For example, use the following:

```
-s "wt.inf.container.SiteOrganization.name=ACME Corporation"
```

On a UNIX system, you can use double quotes or you can escape the space character with a backslash. For example, use the following:

```
-s wt.inf.container.SiteOrganization.name=ACME\ Corporation"
```

On UNIX, dollar signs are usually interpreted by shells as variable prefixes. To set a property value that has a dollar symbol in it, use single quotes around the argument so that it is not interpreted by the shell or use backslash to escape the dollar symbols. For example, use either of the following:

```
-s 'wt.homepage.jsp=$(wt.server.codebase)/wtcore/jsp/wt/portal/  
index.jsp'
```

or

```
-s wt.homepage.jsp=  
\$(wt.server.codebase)/wtcore/jsp/wt/portal/index.jsp
```

Other than escaping arguments so that the command line shell does not misinterpret them, the values should not need to be escaped any further to be compatible with XML or property file syntaxes. The xconfmanager escapes property names and values automatically if necessary.

## About the Windchill Command

PTC has provided a command, `windchill`, to invoke Windchill actions. For example, the command can be used to stop and start Windchill, check the status of the Windchill server, and create a new shell and set the environment variables. It can also be used as a Java wrapper. In that regard, it can accept a Class file as an argument, just like Java, and execute it without a predefined environment (Windchill classes in CLASSPATH, Java in PATH, and so on).

The `windchill` command should be used to execute any server-side Windchill Java code. This will insure that the environment that the command is executed in is properly setup. The environment that actions are executed within, including the `windchill` shell action, is defined by the `wt.env` properties in the `wt.properties` file. For example, the `wt.env.CLASSPATH` property will set the CLASSPATH environment variable for the action that is being invoked.

The `windchill` command is a Perl script that has also been compiled into a Windows binary executable. For UNIX systems, Perl 5.0 or greater must be installed. The `windchill` script assumes that Perl is installed in the standard install location of `/usr/bin/perl`. If Perl is not installed at this location, you can either create a symbolic link (recommended method) to the Perl install location or edit the `windchill` script to reference the Perl install location. To modify the `windchill` script, edit the `<Windchill>/bin/windchill` file. Locate the `#!` entry (for example, `#!/usr/bin/perl -w`) and change the Perl directory to the location where Perl is installed.

The `windchill` command is located in the `<Windchill>/bin` directory. If you receive a command not found message when you execute the `windchill` command, add the `<Windchill>/bin` directory to your PATH environment variable. The syntax of the `windchill` command is:

```
windchill [args] action
```

You can display the help for the `windchill` command by executing `windchill` with the `-h` argument or with no argument.

The following tables list some of the arguments and actions applicable to the `windchill` command. To see a complete list of the arguments, use the report generated from the help (argument).

## windchill Arguments:

Arguments (optional)	Description
- h, --help	Displays help and exits.
-v, --[no]verbose	Explains what is being done when a command is executed. Default is noverbose.
-w, --wthome=DIR	Sets the Windchill home directory. Default is the parent directory containing the windchill script.
--java=JAVA_EXE	The Java executable. Default is the wt.java.cmd variable value specified in the \$WT_HOME/code-base/wt.properties file.
-cp, --classpath=PATH	Java classpath. Default is the wt.java.classpath variable value specified in the \$WT_HOME/code-base/wt.properties file.
--javaargs=JAVAARGS	Java command line arguments.

## windchill Actions

Action	Description
shell	Sets up a Windchill environment in a new instance of the currently running shell.
start	Starts the Windchill server.
stop	Stops the Windchill server.
status	Retrieves the status of the Windchill server.
version	Displays the Windchill install version.

Action	Description
properties <i>&lt;resource&gt;[,...][?key[&amp;key2]...]</i>	Displays the properties as seen by Windchill for the given resource with substitution, etc. executed. It can be limited to a given set of keys.  For example:  windchill properties wt.properties — lists all wt.properties  windchill properties wt.properties?wt.server.codebase — lists server codebase  windchill properties wt.properties?wt.env.* — lists all the environment variables use by windchill shell  windchill properties — with no arguments generates the help report
CLASS [CLASS_ARGS]	Run a Windchill class with optional class arguments. For example:  windchill wt.load.Developer -UAOps

## About the windchill shell

The windchill shell brings up a new command shell, from the parent shell that is setup for the Windchill environment. This includes setting all environment variables defined in wt.env property in the wt.properties file.

To execute the windchill shell, at the command prompt enter the following command:

```
windchill shell
```

When you are finished using the windchill shell, you can exit the shell and return to the parent shell.

PTC recommends running all server-side Windchill applications, tools, and utilities from the windchill shell. Also, you can use the windchill shell to set up your development environment to use javac or Java directly.



# 16

## File-Handling Applets

For more UI flexibility and more robust handling of file upload/download/selection than is available through straight HTML, a basic set of lightweight file-handling applets has been provided. These applets can be used in a variety of HTML-generating architectures, including DCA, JSP, and HTML template processing, and can be used with a variety of Windchill business objects.

<b>Topic</b>	<b>Page</b>
The Three Applets .....	16-2
Advantages and Disadvantages .....	16-2
The File Selection Applet .....	16-3
The Upload Applet .....	16-11
The Download Applet .....	16-31

## The Three Applets

### File selection applet

wt\clients\util\FileChooserDropApplet (combined file drop target and file browser launch button or "hyperlink" which launches single-select or multi-select JFileChooser)

### Content upload applet

wt\clients\util\http\UploadApplet.java

### Content download applet

wt\clients\util\http\DownloadApplet.java

## Advantages and Disadvantages

The primary advantages of these applets compared to straight HTML are:

1. Multiple-file handling - can select/upload/download multiple files in a single operation.
2. Drag-and-drop file selection - can drag and drop one or more files from the desktop or Windows Explorer into a Windchill UI
3. Configurability - can configure upload/download behavior through user preferences.
4. Checksum handling - can persist/compare checksums to avoid unnecessary uploads.
5. Deferred upload and remembered values - can be used to implement multi-page wizard or clerk which maintains the selected upload filepaths when switching between pages, and defers all file uploads until all pages of wizard/clerk are completed.
6. Pre-fillable file selection values - ability to update a previously-uploaded file without needing to re-browse for local filepath.

The primary disadvantages of these applets compared to straight HTML are the necessary install prerequisites for each user.

1. Plug-in - they require the one-time installation of a particular Java plug-in on the client machine.
2. Bootstrap loader - for certain secure server configurations, they require the one-time installation to the client machine of the Windchill bootstrap loader which enables the RMI tunneling necessary for the applet to communicate with the server
3. Java class archive - they require the one-time download of a Java class archive, which creates a noticeable loading delay the first time an applet is accessed from a given client machine. Subsequent visits to the same page are much faster because the class archive is cached locally by the Java plug-in.

## The File Selection Applet

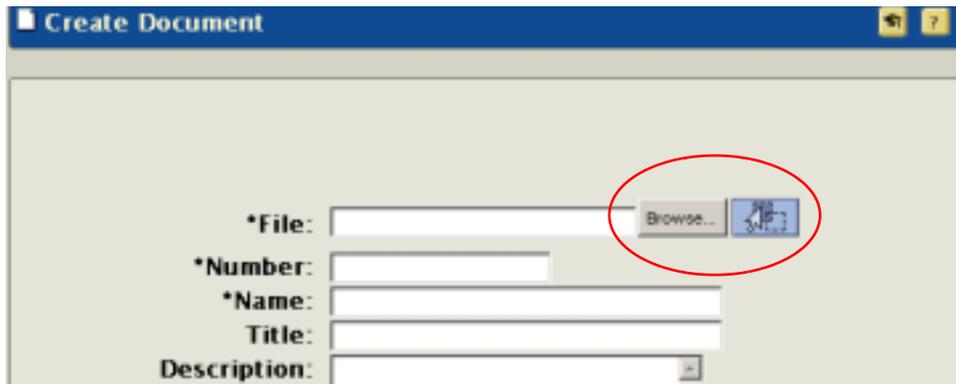
### File Selection

There are two ways to choose one or more files for upload in Windchill.

1. Launch a file browser from the Windchill UI and navigate to the desired file(s).
2. Drag one or more file icons from the desktop or Windows Explorer onto a drop target in the Windchill UI.

### File Browse Launchers

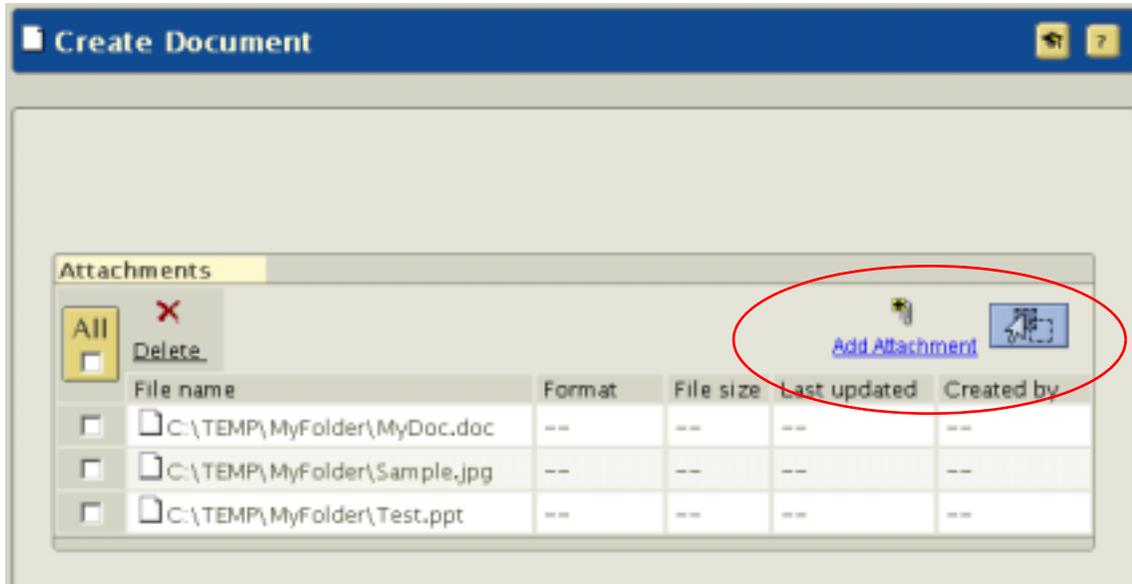
There are two looks for the applet button that launches a Java file browser. The "button" look is typically used in a single-select situation where the selected file will be written to an adjacent HTML textfield, in order to look and act like an HTML file input (only smarter!).



#### Button look:

```
<param name="buttonLook" value="true">
```

The "hyperlink with icon" look is typically used in a multi-select situation where each selected file will result in another row added to a table.



**Hyperlink-with-icon look:**

```
<param name="buttonLook" value="false">  
<param name="imageFilename" value="wtcore/images/attach_add.gif">  
<param name="multiSelect" value="true">
```

**Drop targets**

The presence/absence of a file drop target next to the file browse launcher is determined by the following parameter (true means drop target, false means no drop target):

```
<param name="acceptFileDrop" value="true">
```

When using a drop target, you will need to provide a pair of images to use for the drop target's normal state and active (dragged-over) state.

The following images are provided by Windchill:

**Normal:**

wtcore/images/droptarget.gif



**Active:**

wtcore/images/droptarget\_active.gif



## Processing File Selection Output

The file selection applet will output the selected filepaths to the page by calling a Javascript method of specified signature: `setPath (newValue, fileSep, pathComplete)`. The name of the method may be changed by setting an applet parameter, but the number and type of arguments must conform to the applet's expectations (see examples).

For single-select situations, the Javascript method typically takes the result string from an applet, verifies that only a single file has been selected, then calls another method that writes the filepath to a form field.

For multiple-select situations, the Javascript method typically takes the result string from an applet, then calls another method that performs parsing and/or concatenation on the result string in order to update hidden field values and submit the page to some sort of processor that generates an additional row in the table for each file selected.

Various browser and encoding issues may affect handling of the backslash file separators in Windows filepaths. Some sort of escaping or replacement of these file separators is recommended, such as is performed by the method `CheckFilePathSeparators` in the file templates\doc\WTDocumentSharedFunctions.js.

**Note:** Due to restrictions on the number of characters that can be sent at once, the `FileChooserDropApplet` requires Javascript that supports receiving long filepath strings in multiple chunks. As of 7.0, other file selection applets do not support this ability and are thus limited in the total combined length of filepaths they can return in a single browse/drop.

### Sample single-select Javascript for FileChooserDropApplet:

```
var DELIM = ";;;qqq"; // same as applet's delim parameter
var appletString = "";

function setPathFromApplet(newValue, fileSep, pathComplete) {
    appletString = appletString + newValue;
    if ( pathComplete != null && pathComplete == "true" ) {
        window.focus();
        var endPos = newValue.indexOf( DELIM, 0 );
        if ( endPos >= 0 ) {
```

```

        wfalert( tooManyFilesDroppedMsg );
    } else {
        setPath(CheckFilePathSeparators(appletString,fileSep));
    }
    appletString = "";
}
}

```

### Sample multi-select Javascript for FileChooserDropApplet:

```

var DELIM = ";;;qqq"; // same as applet's delim parameter
var appletString = "";

function setPath(newValue,fileSep,pathComplete) {
    appletString = appletString + newValue;
    if ( pathComplete != null && pathComplete == "true" ) {window.focus();
submitNewAttachments(CheckFilePathSeparators(appletString,fileSep) );appletString =
"";
    }
}

```

## FileChooserDropApplet Parameters

Parameter Name	Sample Value	Description
java_codebase	http://machinename.ptcnet.ptc.com/Windchill/	Provided by taglib generator
type	application/x-java-applet;version=1.4	Provided by taglib generator
java_code	wt/clients/util/FileChooserDropApplet	Applet location in codebase – never changes
cache_archive	"contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar, content3rdParty.jar, content.jar"	Java archives containing the classfiles used by this applet – typically never changes
java_archive	wt/security/security.jar	Never changes
cabinets	wt/security/security.cab	Never changes
cache_option	Plugin	Never changes
SCRIPTABLE	true	Never changes
MAYSCRIPT	true	Never changes
wt.context.locale	en_US	Locale of user's browser
bgcolor	15066585	Decimal version of desired background color (i.e. what is seen around the button), typically matches the background color of the page or table cell where the applet is located
buttonLabel	"Browse... ", "Add Attachments... ", "Add Files... "	Label for button or "hyperlink" seen in HTML page
acceptFileDrop	true, false	Normally true, change to false if you don't want a drop target

dropImageFilename	wtcore/images/droptarget.gif	Image displayed when drop target is idle
dropImageFilenameActive	wtcore/images/droptarget_active.gif	Image displayed when drop target is dragged-over
buttonLook	true, false	Normally true for single-select usage, normally false for multi-select usage
imageFilename	wtcore/images/attach_add.gif	(Optional) Icon shown above "hyperlink" if buttonLook is false
actionLabel	"Open", "Choose File", "Select Attachments"	(Optional) Label for window and action button on file browser dialog
multiSelect	true, false	(Optional) Defaults to false for single-select, change to true for multi-select.
delim	:::zzz	(Optional) Delimiter string used to separate filepaths if multiSelect is true
debug	true, false	(Optional) Defaults to false, change to true for Java Console output during processing

## Sample HTML for FileChooserDropApplet

### FileChooserDropApplet

Create/Update/Checkin Document primary content (single-select, button look)

```
<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFFEDCBA" name="formApplet"
width="200" height="40"
codebase="http://java.sun.com/products/plugin/autodl/jinstall-1_4_2-windows-
i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/FileChooserDropApplet">
<param name="java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
```

```

<param name=" java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="SCRIPTABLE" value="true">
<param name="actionLabel" value="Open">
<param name="debug" value="false">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar,
lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="dropImageFilenameActive" value="wtcore/images/droptarget_active.gif">
<param name="buttonLabel" value="Browse...">
<param name="acceptFileDrop" value="true">
<param name="MAYSCRIPT" value="true">
<param name="buttonLook" value="true">
<param name="cabinets" value="wt/security/security.cab">
<param name="dropImageFilename" value="wtcore/images/droptarget.gif">
<param name="cache_option" value="Plugin">
<param name="wt.context.locale" value="en_US">
<param name="bgcolor" value="15066585">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="200"
height="40" MAYSCRIPT=true pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/FileChooserDropApplet"
java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
SCRIPTABLE="true"
actionLabel="Open"
debug="false"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar,
content3rdParty.jar, content.jar"
dropImageFilenameActive="wtcore/images/droptarget_active.gif"
buttonLabel="Browse..."
acceptFileDrop="true"
buttonLook="true"
cabinets="wt/security/security.cab"
dropImageFilename="wtcore/images/droptarget.gif"
cache_option="Plugin"
wt.context.locale="en_US"
bgcolor="15066585"
><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

### Create/Update Document attachments, Create Multiple Documents (multi-select, hyperlink-with-icon look)

```

<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFEDCBA" name="formApplet"
width="300" height="50"
codebase="http://java.sun.com/products/plugin/autodl/jinstall-1_4_2-windows-
i586.cab#Version=1,4,2">
<param name=" java_code" value="wt/clients/util/FileChooserDropApplet">
<param name=" java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
<param name=" java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="SCRIPTABLE" value="true">

```

```

<param name="actionLabel" value="Open">
<param name="debug" value="false">
<param name="delim" value=";;;zzz">
<param name="multiSelect" value="true">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar,
lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="dropImageFilenameActive" value="wtcore/images/droptarget_active.gif">
<param name="acceptFileDrop" value="true">
<param name="buttonLabel" value="Add+Attachment">
<param name="MAYSCRIPT" value="true">
<param name="imageFilename" value="wtcore/images/attach_add.gif">
<param name="cabinets" value="wt/security/security.cab">
<param name="dropImageFilename" value="wtcore/images/droptarget.gif">
<param name="cache_option" value="Plugin">
<param name="wt.context.locale" value="en_US">
<param name="bgcolor" value="15066585">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="300"
height="50" MAYSCRIPT=true pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/FileChooserDropApplet"
java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
SCRIPTABLE="true"
actionLabel="Open"
debug="false"
delim=";;;zzz"
multiSelect="true"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar,
content3rdParty.jar, content.jar"
dropImageFilenameActive="wtcore/images/droptarget_active.gif"
acceptFileDrop="true"
buttonLabel="Add+Attachment"
imageFilename="wtcore/images/attach_add.gif"
cabinets="wt/security/security.cab"
dropImageFilename="wtcore/images/droptarget.gif"
cache_option="Plugin"
wt.context.locale="en_US"
bgcolor="15066585"
><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

## The Upload Applet

UploadApplet is an "invisible" applet, that is, a 2-pixel by 2-pixel applet whose background color matches the page. It starts the upload as soon as the page has loaded and the applet has initialized, using the values in the applet parameters. This behavior makes UploadApplet most appropriate for use in a "processing" page or popup rather than an interactive form page.

In order to perform an upload, the applet needs to know what content item to upload and an upload destination where it should be uploaded. The item is described by a target type (FILE, URL or NONE) and a target. Items of type FILE have a filepath as a target, items of type URL have a URL string as a target. The upload destination is described by an upload destination URL. The only Windchill-supported use of UploadApplet is to upload content to an upload URL that leads to the saveContent method of the wt.content.ContentHttp class in the Windchill Method Server.

**Caution:** While it may be possible to use UploadApplet to upload content to some other type of upload URL, such as an Info\*Engine URL or even a totally non-Windchill URL, any such "off-label" use of UploadApplet is NOT considered supported by Windchill's tech support or R & D.

### Initial upload of primary and/or secondary content to a single Windchill object

When initially uploading to a Windchill object, such as during a Create Document or Create Document Template, or when updating a Windchill object which has not previously had content, there is no existing Windchill content item yet in the database so a new content item object will have to be created. This is indicated for primary content by setting the callingAction parameter to "create", and for secondary content by indicating "newfile" in place of the OID within the concatenated attachments parameter.

### Initial upload of multiple primary files to multiple Windchill objects

When creating multiple documents, multiple primary files can be uploaded to multiple destinations with a single instance of UploadApplet. This is done by setting the "target" parameter to a concatenated list of target filepaths, and setting the "uploadURL" parameter to a corresponding concatenated list of uploadURLs corresponding in quantity and order to the concatenated "target" values.

### Subsequent upload to update/replace/remove content of a Windchill object

When uploading to a single Windchill object that already has content, the about-to-be-uploaded primary file needs to be validated and compared to the currently-persisted primary content item, to avoid unnecessary uploads. This is indicated by setting the callingAction parameter to "updateReplace" and providing persisted filename and checksum information about the existing content item. The result of the validation/comparison, along with certain user preference values, determines whether to upload the file, skip the upload, or prompt the user for an upload decision or different filepath. (Secondary content is validated but no comparisons are performed.)

Existing content items, both primary and secondary, need to have their OID provided so that the upload modifies the content of the existing content item rather than creating an additional content item. When removing primary or secondary content, the OID needs to be provided so that the correct existing content item can be removed. This is indicated for primary content by setting the `oidString` parameter to the OID of the current primary content item, and for secondary content by providing the OID of the to-be-removed-or-replaced attachment content item within the concatenated `attachments` parameter.

**Caution:** The OID must be in the format of `"wt.content.ApplicationData:1437923"`. Attempting to use an OID with a different format, such as `"OR:wt.content.ApplicationData:1437923"`, will result in server-side failure to persist the uploaded object.

### Handling success/error navigation and feedback on uploads

To enable `UploadApplet` to be used in a variety of architectures/situations but maintain feedback behavior and language appropriate to each usage, `UploadApplet` allows the parameters and the page Javascript to define the appropriate feedback display handling and post-upload navigation.

When uploading to a single location, the selected primary content filepath is validated, preference values are checked and the user is prompted if necessary to decide what to do should the selected filepath be invalid. If the file is valid or the preferences and/or user prompt authorize skipping the upload, then the upload is considered successful, and the applet will forward to the `"completionUrl"` and/or `"completionJSMETHOD"` upon completion (if both are provided, `completionUrl` will be passed into `completionJSMETHOD` as an argument value). In Windchill PDM, this typically submits to post-processing code followed by the properties page for the document. In `PDMLink`, this typically submits to post-processing code followed by closing the wizard window. In the absence of both `completionUrl` and `completionJSMETHOD` parameter values, `completionJSMETHOD` will default to `"displayInvalidPaths"`.

**Note:** The `"uploadFeedback"` argument provided to the `completionJSMETHOD` will be a value from the class `wt.clients.util.http.UploadConstants` indicating whether some kind of non-upload completion was reached (`NO_UPLOAD`, `FILE_NOT_FOUND`, `FILE_UNCHANGED`, `DECLINED_UPLOAD`). This can be used for customizing a feedback status message for a non-error outcome.

If the file is invalid and the preferences and/or user prompt do not authorize skipping the upload, or if the file is valid but the upload is unsuccessful, or if the user cancels, then the applet will forward to the `"failureUrl"` and/or `"failureJSMETHOD"` (if both are provided, `failureUrl` will be passed into `failureJSMETHOD` as an argument value). This typically displays some sort of error feedback message (unless cancelled) and returns to the form where the primary filepath was originally selected.

When performing a multiple-file upload, such as multiple primary files or attachments, individual feedback is not provided on each file. Instead, all valid files are uploaded and a list of all invalid filepaths is compiled. When the upload is complete, this list of invalid filepaths is sent as an argument to a designated Javascript method, along with the appropriate URL. When uploading multiple primary files, this will always be the Javascript method and URL specified in the applet parameters for the completion case. When uploading secondary files, the method and URL may be either from either the completion or failure cases, depending on the outcome of the selected primary content file upload.

### Sample Javascript for UploadApplet feedback and navigation:

**Note:** This Javascript assumes the existence of a hidden HTML form in the page named <form name>, containing named hidden fields <checksum form field name> and <uploadFeedback form field name>. The real names of this form and these fields should be substituted into this Javascript before use.

```
<SCRIPT language=javascript>

// localized display messages retrieved from resource bundles
var invalidPathMessage = "The following files were not uploaded due to invalid
filepaths:";
var uploadFailureMessage = "A problem occurred while uploading content to
server. Please notify administrator if this problem continues.";

function uploadCompleted( invalidPaths, nextURL, newChecksum, uploadFeedback ) {
    document.<form name>.<checksum form field name>.value = newChecksum;
    if ( uploadFeedback != null && uploadFeedback.length > 0 ) {
        \document.<form name>.<uploadFeedback form field name>.value =
uploadFeedback;
    }
    if ( invalidPaths != null && invalidPaths.length > 0 ) {
        alert( invalidPathMessage + invalidPaths );
    }
    if ( nextURL != null && nextURL.length > 0 && nextURL != "null" ) {
        submitForm( nextURL );
    } else {
        submitForm( propertiesPageURL ); // generated value
    }
}

function uploadFailed( invalidPaths, nextURL, cancelled ) {
    if ( invalidPaths != null && invalidPaths.length > 0 ) {
        alert( invalidPathMessage + invalidPaths );
    }

    if ( !cancelled ) {
        alert( uploadFailureMessage );
    }

    if ( nextURL != null && nextURL.length > 0 && nextURL != "null" ) {
        submitForm( nextURL );
    }
}
```

```

    } else {
        submitForm( previousPageURL ); // generated value
    }
}

function displayInvalidPaths( invalidPaths, nextURL ) {
    alert( invalidPathMessage + invalidPaths );
    location = nextURL;
}

</SCRIPT>

```

## Persisting Information After an Upload

When uploading to a single document, the primary file checksum should be persisted after a successful upload, for use in future upload comparisons. To minimize network traffic, the checksum is appended to the end of the completionURL query string so that it can be processed in the same server hit as the request for the completion page. The server-side delegate or JSP that handles the completionURL request should check whether the "uploaded" flag on the query string is set to "true", and if so should persist the "checksum" query string value to the context object's primary content item.

### Checksum persistence code:

Here are the essential steps for persisting the checksum (in reality, there may also be some error handling). The context object has to be a FormatContentHolder, such as a WTDocument.

```

Properties props = getQueryData();
String uploaded = (String)props.get("uploaded");
if ( uploaded != null && uploaded.equalsIgnoreCase("true") ) {
    long checksum = 0;
    checksum = Long.parseLong((String)props.get("checksum"));
    FormatContentHolder holder =
    (FormatContentHolder)getContextObj();
    holder =
    (FormatContentHolder)ContentHelper.service.getContents(holder);
    ContentItem item = ContentHelper.getPrimary( holder );

    if ( item instanceof ApplicationData ) {
        ApplicationData app = (ApplicationData)item;
        app.setChecksum( checksum );
        ContentHelper.service.updateAppData( holder, app );
    }
}

```

## Configuring Upload Behavior - System Properties

The optional "validEmptyFile" and "uploadImpl" applet parameters are typically set by retrieving the wt.properties values for "wt.content.validEmptyFile" and "wt.content.uploadImpl", respectively.

The `validEmptyFile` parameter allows a particular implementation to determine whether or not a 0-size "empty" file is considered a valid content file or not - the default value is "false", that an empty file is not considered valid. This value typically will not be changed unless Windchill is being used to automate data storage from some other application that generates 0-size placeholder files.

The `uploadImpl` parameter permits switching between RMI and HTTP upload mechanisms - the default value is "rmi" which provides full-functionality Windchill upload including full support of file vault caching and content replication. The other alternative is "http" which permits uploading to a non-Windchill URL but has severe limitations in both functionality and performance. This value typically will not be changed at the `wt.properties` level due to loss of functionality, but can be overridden at the page level by setting the applet parameter to allow an isolated unsupported instance of `UploadApplet` uploading to a non-RMI upload URL, without losing the RMI upload functionality in all other instances of `UploadApplet`.

**Caution:** Changing the value of the `wt.content.uploadImpl` `wt.properties` value or `uploadImpl` applet parameter is possible but is NOT supported - use at your own risk.

### Configuring upload behavior - user preferences

The applet knows how to compare the new primary content filepath with the previously-persisted filename/checksum, and how to make sure the new filepath is valid, but the resulting behavior is determined by parameters that are typically set by the individual user's preferences.

The Changed File Behavior preference (node=`/wt/content` key=`uploadIfFileChanged`) determines whether changes to the primary file are automatically uploaded, or whether the user wants to have the option to accept the upload or not. Most users will prefer to automatically upload a changed file without prompting, but some users who frequently do updates and checkins for metadata or attachments reasons might prefer the ability to be prompted in case the changes they're making to the primary content aren't yet ready to be checked in for others to see.

The File Not Found Behavior preference (node=`/wt/content` key=`continueIfFileNotFound`) determines whether an invalid filepath means a failure condition or not. Most users will prefer to be notified of an invalid filepath so that they can correct the selection, but some users who frequently do updates and checkins for metadata or attachments reasons might set their download preferences to automatically skip the primary content download on checkout and set this upload preference to automatically skip the upload on update/checkin if the primary content file doesn't exist locally.

The Unchanged File Behavior preference (node=`/wt/content` key=`continueIfFileUnchanged`) determines whether the absence of changes to the primary file automatically means to skip the upload as unnecessary, or whether the user wants to be notified and have the option to skip the upload or select a

different primary file. Most users will only have a single local copy of the primary file and thus will prefer to skip the upload if the primary content hasn't changed, but users who routinely edit a copy of the primary content file in a different location or under a different filename might want to have the option to be reminded in case they accidentally select the original file rather than the changed file.

## Upload Applet Parameters

Parameter Name	Sample Value	Description
java_codebase	http://machinename.ptcnet.ptc.com/Windchill /	Provided by taglib generator
type	application/x-java-applet;version=1.4	Provided by taglib generator
java_code	wt/clients/util/UploadApplet	Applet location in codebase – never changes
cache_archive	"contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar, content3rdParty.jar, content.jar"	Java archives containing the classfiles used by this applet
java_archive	wt/security/security.jar	Never changes
cabinets	wt/security/security.cab	Never changes
cache_option	Plugin	Never changes
SCRIPTABLE	true	Never changes
MAYSCRIPT	true	Never changes
wt.context.locale	en_US	Locale of user's browser
bgcolor	16777215	Decimal version of desired background color (i.e. what is seen around the button), typically matches the background color of the page or table cell where the applet is located
removable	true	Never changes
target	C:/Temp/MyFolder/Sample.jpg	Local filepath(s) to the new primary content item(s) to be uploaded
targetType	FILE, URL, NONE	Category of newly-selected primary content
uploadURL	http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.content.ContentHttp/saveContent?wt.doc.WTDocument%3A656270	URL(s) to accept the upload, typically leads to saveContent method of wt.content.ContentHttp in the Windchill Method Server

docOperation	create, update	Refers to context document – create if this upload takes place as part of the initial creation of the document, update if this is a subsequent update or checkin
callingAction	create, updateReplace	Refers to primary content item - create if there is currently no persisted primary content item, updateReplace if there is already a persisted primary content item
oidString	wt.content.ApplicationData:656261	On updateReplace, the OID of the persisted primary content item
attachments	FILE;;;qqqnewfile;;;qqqC:\TEMP\DgadReq.doc;;;qqqADD;;;zzzURL;;;qqqnewURL;;;qqqwww.amazon.com;;;qqq;;;qqqADD;;;zzzFILE;;;qqqwt.content.ApplicationData:656293;;;qqqC:\TEMP\MyDoc.doc;;;qqqREPLACE;;;zzzURL;;;qqqwt.content.URLData:656295;;;qqqhttp://www.askjeeves.com;;;qqqAsk Jeeves website;;;qqqREPLACE;;;zzzFILE;;;qqqwt.content.ApplicationData:656902;;;qqq;;;qqqREMOVE;;;zzzURL;;;qqqwt.content.URLData:656904;;;qqq;;;qqqREMOVE	Type, content item OID (if any), filepath/URL, description (if any) and action for any attachments to be added, removed or replaced. Attachments are separated by contentRecordDelim value, individual fields for each attachment are separated by contentDelim value
multipleQuantity	3	(Optional) Number of different primary files/destinations
contentDelim	;;;qqq	(Optional) Delimiter within attachments entries and between multiple-upload targets/uploadURLs, defaults to ;;;qqq
contentRecordDelim	;;;zzz	(Optional) Delimiter used between separate attachments entries, defaults to ;;;zzz
completionJSMethode	uploadCompleted	(Optional) Methodname of Javascript method to be called if upload doesn't error out

completionUrl	"http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.enterprise.URLProcessor/invokeAction?action=CreateDocCloseWindow&class=wt.doc.WTDocument&OID=VR%3Awt.doc.WTDocument%3A288876&refresh=true&formName=CreateDocumentWizard">	(Optional) URL to follow if upload doesn't error out
failureJSMethod	uploadFailed	(Optional) Methodname of Javascript method to be called if upload errors out
failureUrl	"http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.enterprise.URLProcessor/invokeAction?nextAction=CreateDocument2&action=CreateDocumentUploadFailure&class=wt.doc.WTDocument&OID=VR%3Awt.doc.WTDocument%3A288876&uploaded=false&formName=CreateDocumentWizard"	(Optional) URL to follow if upload errors out
checksum	3774325531	(Optional on updateReplace) The checksum from previous upload of persisted primary content item
fileName	MyDoc.doc	(Optional on updateReplace) The filename of the persisted primary content item
uploadedFromPath	C:/temp/MyDoc.doc	(Optional on updateReplace) The filepath from previous upload of persisted primary content item
continueIfFileNotFound	true, false	(Optional on updateReplace) Determines whether to automatically skip the upload and keep current primary content if the selected file cannot be found, normally retrieved from user preference setting

continueIfFileUnchanged	true, false	(Optional on updateReplace) Determines whether to automatically skip the upload and keep current primary content if the selected file has not changed since the previous upload, normally retrieved from user preference setting
uploadIfFileChanged	true, false	(Optional on updateReplace) determines whether to automatically upload the file if it has changed since the previous upload, normally retrieved from user preference setting
workspacePath	C:/temp/MyWindchillFiles/dummy.txt	(Optional) Normally retrieved from user preference setting
validEmptyFile	true, false	(Optional) Normally false, change to true if you want to permit 0-size files to be considered valid Windchill content
uploadImpl	rmi, http	(Optional) Defaults to rmi, change to http only for unsupported uses of this applet.
debug	true, false	(Optional) Defaults to false, change to true for Java Console output during processing

## Sample HTML for UploadApplet

### Create Document (add primary/attachment files to single upload destination)

```
<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFFEDCBA" name="formApplet"
width="2" height="2" codebase="http://java.sun.com/products/plugin/autodl/jinstall-
1_4_2-windows-i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/http/UploadApplet">
<param name="java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
<param name="java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="bgcolor" value="16777215">
<param name="validEmptyFile" value="false">
<param name="oid" value="">
<param name="SCRIPTABLE" value="true">
<param name="callingAction" value="create">
<param name="cache_option" value="Plugin">
<param name="MAYSCRIPT" value="true">
<param name="checksum" value="">
<param name="contentRecordDelim" value=";;;zzz">
<param name="continueIfFileNotFound" value="false">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar,
lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="uploadedFromPath" value="">
<param name="completionUrl"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.enterp
rise.URLProcessor/invokeAction?action=CreateDocCloseWindow&class=wt.doc.WTDocument&
OID=VR%3Awt.doc.WTDocument%3A288876&refresh=true&formName=CreateDocumentWizard">
<param name="workspacePath" value="">
<param name="docOperation" value="create">
<param name="debug" value="false">
<param name="failureUrl"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.enterp
rise.URLProcessor/invokeAction?nextAction=CreateDocument2&action=CreateDocumentUplo
adFailure&class=wt.doc.WTDocument&OID=VR%3Awt.doc.WTDocument%3A288876&uploaded=fals
e&formName=CreateDocumentWizard">
<param name="uploadIfFileChanged" value="true">
<param name="targetType" value="">
<param name="cabinets" value="wt/security/security.cab">
<param name="continueIfFileUnchanged" value="true">
<param name="oidString" value="">
<param name="contentDelim" value=";;;qqq">
<param name="wt.context.locale" value="en_US">
<param name="delim" value=";;;qqq">
<param name="uploadURL"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.conten
t.ContentHttp/saveContent?wt.doc.WTDocument%3A288877">
<param name="removable" value="true">
<param name="attachments" value="FILE;;;qqqnewFile;;;qqqC:\TEMP\MyFolder\New
Microsoft Excel Worksheet.xls;;;qqqADD;;;zzzFILE;;;qqqnewFile;;;qqqC:\TEMP\
MyFolder\New Microsoft PowerPoint
Presentation.ppt;;;qqqADD;;;zzzFILE;;;qqqnewFile;;;qqqC:\TEMP\MyFolder\New
Microsoft Word Document.doc;;;qqqADD;;;zzzFILE;;;qqqnewFile;;;qqqC:\TEMP\MyFolder\
New Text Document.txt;;;qqqADD;;;zzzFILE;;;qqqnewFile;;;qqqC:\TEMP\MyFolder\New
WinZip File.zip;;;qqqADD">
<param name="fileName" value="">
```

```

<param name="target" value="c:\temp\MyDoc.doc">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="2"
height="2" MAYSCRIPT=true pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/http/UploadApplet"
java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
bgcolor="16777215"
validEmptyFile="false"
oid=" "
SCRIPTABLE="true"
callingAction="create"
cache_option="Plugin"
checksum=" "
contentRecordDelim=";;;zzz"
continueIfFileNotFound="false"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar,
content3rdParty.jar, content.jar"
uploadedFromPath=" "
completionUrl="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.
enterprise.URLProcessor/invokeAction?action=CreateDocCloseWindow&class=wt.doc.WTD
ocument&OID=VR%3Awt.doc.WTDocument%3A288876&refresh=true&formName=CreateDocumentWiz
ard"
workspacePath=" "
docOperation="create"
debug="false"
failureUrl="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.e
nterprise.URLProcessor/invokeAction?nextAction=CreateDocument2&action=CreateDocumen
tUploadFailure&class=wt.doc.WTDocument&OID=VR%3Awt.doc.WTDocument%3A288876&uploaded
=false&formName=CreateDocumentWizard"
uploadIfFileChanged="true"
targetType=" "
cabinets="wt/security/security.cab"
continueIfFileUnchanged="true"
oidString=" "
contentDelim=";;;qqq"
wt.context.locale="en_US"
delim=";;;qqq"
uploadURL="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.co
ntent.ContentHttp/saveContent?wt.doc.WTDocument%3A288877"
removable="true"
attachments="FILE;;;qqqnewFile;;;qqqC:\TEMP\MyFolder\New Microsoft Excel
Worksheet.xls;;;qqqADD;;;zzzFILE;;;qqqnewFile;;;qqqC:\TEMP\MyFolder\New Microsoft
PowerPoint Presentation.ppt;;;qqqADD;;;zzzFILE;;;qqqnewFile;;;qqqC:\TEMP\MyFolder\
New Microsoft Word Document.doc;;;qqqADD;;;zzzFILE;;;qqqnewFile;;;qqqC:\TEMP\
MyFolder\New Text Document.txt;;;qqqADD;;;zzzFILE;;;qqqnewFile;;;qqqC:\TEMP\
MyFolder\New WinZip File.zip;;;qqqADD"
fileName=" "
target="c:\temp\MyDoc.doc"
><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

## Create Document (add primary/attachment URLs to single upload destination)

```
<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFEDCBA" name="formApplet"
width="2" height="2" codebase="http://java.sun.com/products/plugin/autodl/jinstall-
1_4_2-windows-i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/http/UploadApplet">
<param name="java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
<param name="java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="bgcolor" value="16777215">
<param name="validEmptyFile" value="false">
<param name="oid" value="">
<param name="SCRIPTABLE" value="true">
<param name="callingAction" value="create">
<param name="cache_option" value="Plugin">
<param name="MAYSCRIPT" value="true">
<param name="checksum" value="">
<param name="contentRecordDelim" value=";;;zzz">
<param name="continueIfFileNotFound" value="false">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar,
lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="completionJSMETHOD" value="uploadCompleted">
<param name="uploadedFromPath" value="">
<param name="completionUrl" value="">
<param name="workspacePath" value="">
<param name="docOperation" value="create">
<param name="failureJSMETHOD" value="uploadFailed">
<param name="debug" value="false">
<param name="failureUrl" value="">
<param name="uploadIfFileChanged" value="true">
<param name="targetType" value="URL">
<param name="cabinets" value="wt/security/security.cab">
<param name="continueIfFileUnchanged" value="true">
<param name="oidString" value="">
<param name="contentDelim" value=";;;qqq">
<param name="wt.context.locale" value="en_US">
<param name="delim" value=";;;qqq">
<param name="uploadURL"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.conten
t.ContentHttp/saveContent?wt.doc.WTDocument%3A656270">
<param name="removable" value="true">
<param name="fileName" value="">
<param name="attachments" value="URL;;;qqqnewURL;;;qqqwww.google.com;;;qqqGoogle
search engine;;;qqqADD;;;zzzURL;;;qqqnewURL;;;qqqwww.cnn.com;;;qqq;;;qqqADD">
<param name="target" value="www.ptc.com">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="2"
height="2" MAYSCRIPT=true pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/http/UploadApplet"
java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
bgcolor="16777215"
validEmptyFile="false"
oid=""
SCRIPTABLE="true"
callingAction="create"
cache_option="Plugin"
```

```

checksum=" "
contentRecordDelim=";;;zzz"
continueIfFileNotFound="false"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar,
content3rdParty.jar, content.jar"
completionJSMETHOD="uploadCompleted"
uploadedFromPath=" "
completionUrl=" "
workspacePath=" "
docOperation="create"
failureJSMETHOD="uploadFailed"
debug="false"
failureUrl=" "
uploadIfFileChanged="true"
targetType="URL"
cabinets="wt/security/security.cab"
continueIfFileUnchanged="true"
oidString=" "
contentDelim=";;;qqq"
wt.context.locale="en_US"
delim=";;;qqq"
uploadURL="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.co
ntent.ContentHttp/saveContent?wt.doc.WTDocument%3A656270"
removable="true"
fileName=" "
attachments="URL;;;qqqnewURL;;;qqqwww.google.com;;;qqqGoogle search
engine;;;qqqADD;;;zzzURL;;;qqqnewURL;;;qqqwww.cnn.com;;;qqq;;;qqqADD"
target="www.ptc.com"
<NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

### Create Multiple Documents (add primary files to multiple upload destinations)

```

<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFEDCBA" width="2" height="2"
codebase="http://java.sun.com/products/plugin/autodl/jinstall-1_4_2-windows-
i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/http/UploadApplet.class">
<param name="java_codebase" value="http://lqkohpsh.ptcnet.ptc.com:2801/PDMLink700">
<param name="java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="bgcolor" value="16777215">
<param name="oid" value=" ">
<param name="SCRIPTABLE" value="true">
<param name="multipleQuantity" value="3">
<param name="callingAction" value="create">
<param name="cache_option" value="Plugin">
<param name="type" value="application/x-java-applet">
<param name="MAYSCRIPT" value="true">
<param name="contentRecordDelim" value=";;;zzz">
<param name="continueIfFileNotFound" value="false">
<param name="cache_archive" value="lib/HTTPClient.jar, contentCust.jar,
contentDSU.jar, contentFCS.jar, content3rdParty.jar, content.jar">
<param name="jreversion" value="1.4">
<param name="completionJSMETHOD" value="completion">

```

```

<param name="uploadedFromPath" value="">
<param name="workspacePath" value="">
<param name="failureJSMethod" value="failure">
<param name="debug" value="true">
<param name="java_archive" value="wt/security/security.jar">
<param name="uploadIfFileChanged" value="true">
<param name="targetType" value="">
<param name="cabinets" value="wt/security/security.cab">
<param name="continueIfFileUnchanged" value="true">
<param name="java_code" value="wt/clients/util/http/UploadApplet">
<param name="contentDelim" value=";;;qqq">
<param name="wt.context.locale" value="en_US">
<param name="delim" value=";;;qqq">
<param name="uploadURL"
value="http://lqkohpsh.ptcnet.ptc.com:2801/PDMLink700/servlet/WindchillAuthGW/wt.co
ntent.ContentHttp/saveContent?wt.doc.WTDocument%3A20350;;;qqqhttp://lqkohpsh.ptcnet
.ptc.com:2801/PDMLink700/servlet/WindchillAuthGW/wt.content.ContentHttp/saveContent
?wt.doc.WTDocument%3A20335;;;qqqhttp://lqkohpsh.ptcnet.ptc.com:2801/PDMLink700/serv
let/WindchillAuthGW/wt.content.ContentHttp/saveContent?wt.doc.WTDocument%3A20365">
<param name="removable" value="true">
<param name="attachments" value="">
<param name="target" value="c:\temp\myfolder\New Text Document.txt;;;qqqc:\temp\
myfolder\New Bitmap Image.bmp;;;qqqc:\temp\myfolder\New WinZip File.zip">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" width="2" height="2"
MAYSCRIPT=true pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/http/UploadApplet.class"
java_codebase="http://lqkohpsh.ptcnet.ptc.com:2801/PDMLink700"
java_archive="wt/security/security.jar"
bgcolor="16777215"
oid=""
SCRIPTABLE="true"
multipleQuantity="3"
callingAction="create"
cache_option="Plugin"
type="application/x-java-applet"
contentRecordDelim=";;;zzz"
continueIfFileNotFound="false"
cache_archive="lib/HTTPClient.jar, contentCust.jar, contentDSU.jar, contentFCS.jar,
content3rdParty.jar, content.jar"
jreversion="1.4"
completionJSMethod="completion"
uploadedFromPath=""
workspacePath=""
failureJSMethod="failure"
debug="true"
java_archive="wt/security/security.jar"
uploadIfFileChanged="true"
targetType=""
cabinets="wt/security/security.cab"
continueIfFileUnchanged="true"
java_code="wt/clients/util/http/UploadApplet"
contentDelim=";;;qqq"
wt.context.locale="en_US"
delim=";;;qqq"

```

```

uploadURL="http://lqkohpsh.ptcnet.ptc.com:2801/PDMLink700/servlet/WindchillAuthGW/w
t.content.ContentHttp/saveContent?wt.doc.WTDocument%3A20350;;;qqqhttp://lqkohpsh.pt
cnet.ptc.com:2801/PDMLink700/servlet/WindchillAuthGW/wt.content.ContentHttp/saveCon
tent?wt.doc.WTDocument%3A20335;;;qqqhttp://lqkohpsh.ptcnet.ptc.com:2801/PDMLink700/
servlet/WindchillAuthGW/wt.content.ContentHttp/saveContent?wt.doc.WTDocument%3A2036
5"
removable="true"
attachments=""
target="c:\temp\myfolder\New Text Document.txt;;;qqqc:\temp\myfolder\New Bitmap
Image.bmp;;;qqqc:\temp\myfolder\New WinZip File.zip"
<NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

## Update Document (replace primary file)

```

<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFFEDCBA" name="formApplet"
width="2" height="2" codebase="http://java.sun.com/products/plugin/autodl/jinstall-
1_4_2-windows-i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/http/UploadApplet">
<param name="java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
<param name="java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="bgcolor" value="16777215">
<param name="validEmptyFile" value="false">
<param name="oid" value="VR:wt.doc.WTDocument:656218">
<param name="SCRIPTABLE" value="true">
<param name="callingAction" value="updateReplace">
<param name="cache_option" value="Plugin">
<param name="MAYSCRIPT" value="true">
<param name="checksum" value="3774325531">
<param name="contentRecordDelim" value=";;;zzz">
<param name="continueIfFileNotFound" value="false">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar,
lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="completionJSMMethod" value="uploadCompleted">
<param name="uploadedFromPath" value="c:/temp/MyDoc.doc">
<param name="completionUrl" value="">
<param name="workspacePath" value="">
<param name="docOperation" value="update">
<param name="failureJSMMethod" value="uploadFailed">
<param name="debug" value="false">
<param name="failureUrl" value="">
<param name="uploadIfFileChanged" value="true">
<param name="targetType" value="FILE">
<param name="cabinets" value="wt/security/security.cab">
<param name="continueIfFileUnchanged" value="true">
<param name="oidString" value="wt.content.ApplicationData:656261">
<param name="contentDelim" value=";;;qqq">
<param name="wt.context.locale" value="en_US">
<param name="delim" value=";;;qqq">
<param name="uploadURL"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.conten
t.ContentHttp/saveContent?wt.doc.WTDocument%3A656220">
<param name="removable" value="true">

```

```

<param name="fileName" value="MyDoc.doc">
<param name="attachments" value="">
<param name="target" value="C:/TEMP/MyFolder/Sample.jpg">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="2"
height="2" MAYSCRIPT=true pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/http/UploadApplet"
java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
bgcolor="16777215"
validEmptyFile="false"
oid="VR:wt.doc.WTDocument:656218"
SCRIPTABLE="true"
callingAction="updateReplace"
cache_option="Plugin"
checksum="3774325531"
contentRecordDelim=";;;zzz"
continueIfFileNotFound="false"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar,
content3rdParty.jar, content.jar"
completionJSMMethod="uploadCompleted"
uploadedFromPath="c:/temp/MyDoc.doc"
completionUrl=""
workspacePath=""
docOperation="update"
failureJSMMethod="uploadFailed"
debug="false"
failureUrl=""
uploadIfFileChanged="true"
targetType="FILE"
cabinets="wt/security/security.cab"
continueIfFileUnchanged="true"
oidString="wt.content.ApplicationData:656261"
contentDelim=";;;qqq"
wt.context.locale="en_US"
delim=";;;qqq"
uploadURL="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.co
ntent.ContentHttp/saveContent?wt.doc.WTDocument%3A656220"
removable="true"
fileName="MyDoc.doc"
attachments=""
target="C:/TEMP/MyFolder/Sample.jpg"
><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

### Update Document (remove primary file)

```

<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFEDCBA" name="formApplet"
width="2" height="2" codebase="http://java.sun.com/products/plugin/autodl/jinstall-
1_4_2-windows-i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/http/UploadApplet">
<param name="java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
<param name="java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">

```

```

<param name="bgcolor" value="16777215">
<param name="validEmptyFile" value="false">
<param name="oid" value="VR:wt.doc.WTDocument:656218">
<param name="SCRIPTABLE" value="true">
<param name="callingAction" value="updateReplace">
<param name="cache_option" value="Plugin">
<param name="MAYSCRIPT" value="true">
<param name="checksum" value="3774325531">
<param name="contentRecordDelim" value=";;;zzz">
<param name="continueIfFileNotFound" value="false">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar,
lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="completionJSMMethod" value="uploadCompleted">
<param name="uploadedFromPath" value="c:/temp/MyDoc.doc">
<param name="completionUrl" value="">
<param name="workspacePath" value="">
<param name="docOperation" value="update">
<param name="failureJSMMethod" value="uploadFailed">
<param name="debug" value="false">
<param name="failureUrl" value="">
<param name="uploadIfFileChanged" value="true">
<param name="targetType" value="NONE">
<param name="cabinets" value="wt/security/security.cab">
<param name="continueIfFileUnchanged" value="true">
<param name="oidString" value="wt.content.ApplicationData:656261">
<param name="contentDelim" value=";;;qqq">
<param name="wt.context.locale" value="en_US">
<param name="delim" value=";;;qqq">
<param name="uploadURL"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.conten
t.ContentHttp/saveContent?wt.doc.WTDocument%3A656220">
<param name="removable" value="true">
<param name="fileName" value="MyDoc.doc">
<param name="attachments" value="">
<param name="target" value="">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="2"
height="2" MAYSCRIPT=true pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/http/UploadApplet"
java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
bgcolor="16777215"
validEmptyFile="false"
oid="VR:wt.doc.WTDocument:656218"
SCRIPTABLE="true"
callingAction="updateReplace"
cache_option="Plugin"
checksum="3774325531"
contentRecordDelim=";;;zzz"
continueIfFileNotFound="false"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar,
content3rdParty.jar, content.jar"
completionJSMMethod="uploadCompleted"
uploadedFromPath="c:/temp/MyDoc.doc"
completionUrl=""
workspacePath=""
docOperation="update"

```

```

failureJSMETHOD="uploadFailed"
debug="false"
failureUrl=""
uploadIfFileChanged="true"
targetType="NONE"
cabinets="wt/security/security.cab"
continueIfFileUnchanged="true"
oidString="wt.content.ApplicationData:656261"
contentDelim=";;;qqq"
wt.context.locale="en_US"
delim=";;;qqq"
uploadURL="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.co
ntent.ContentHttp/saveContent?wt.doc.WTDocument%3A656220"
removable="true"
fileName="MyDoc.doc"
attachments=""
target=""
><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

### Update Document (no primary content handling, add/replace/remove attachment files/URLs)

```

<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFEDCBA" name="formApplet"
width="2" height="2" codebase="http://java.sun.com/products/plugin/autodl/jinstall-
1_4_2-windows-i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/http/UploadApplet">
<param name="java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
<param name="java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="bgcolor" value="16777215">
<param name="validEmptyFile" value="false">
<param name="oid" value="VR:wt.doc.WTDocument:656269">
<param name="SCRIPTABLE" value="true">
<param name="callingAction" value="create">
<param name="cache_option" value="Plugin">
<param name="MAYSCRIPT" value="true">
<param name="checksum" value="">
<param name="contentRecordDelim" value=";;;zzz">
<param name="continueIfFileNotFound" value="false">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar,
lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="completionJSMETHOD" value="uploadCompleted">
<param name="uploadedFromPath" value="">
<param name="completionUrl" value="">
<param name="workspacePath" value="">
<param name="docOperation" value="update">
<param name="failureJSMETHOD" value="uploadFailed">
<param name="debug" value="false">
<param name="failureUrl" value="">
<param name="uploadIfFileChanged" value="true">
<param name="targetType" value="NONE">
<param name="cabinets" value="wt/security/security.cab">
<param name="continueIfFileUnchanged" value="true">

```

```

<param name="oidString" value="">
<param name="contentDelim" value=";;qqq">
<param name="wt.context.locale" value="en_US">
<param name="delim" value=";;qqq">
<param name="uploadURL"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.conten
t.ContentHttp/saveContent?wt.doc.WTDocument%3A656270">
<param name="removable" value="true">
<param name="fileName" value="">
<param name="attachments" value="FILE;;;qqqnewfile;;;qqqC:\TEMP\
DGadReq.doc;;;qqqADD;;;zzzURL;;;qqqnewURL;;;qqqwww.amazon.com;;;qqq;;;qqqADD;;;zzzF
ILE;;;qqqwt.content.ApplicationData:656293;;;qqqC:\TEMP\
MyDoc.doc;;;qqqREPLACE;;;zzzURL;;;qqqwt.content.URLData:656295;;;qqqhttp://www.askj
eeves.com;;;qqqAsk Jeeves
website;;;qqqREPLACE;;;zzzFILE;;;qqqwt.content.ApplicationData:656902;;;qqq;;;qqqRE
MOVE;;;zzzURL;;;qqqwt.content.URLData:656904;;;qqq;;;qqqREMOVE">
<param name="target" value="">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="2"
height="2" MAYSCRIPT=true pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/http/UploadApplet"
java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
bgcolor="16777215"
validEmptyFile="false"
oid="VR:wt.doc.WTDocument:656269"
SCRIPTABLE="true"
callingAction="create"
cache_option="Plugin"
checksum=""
contentRecordDelim=";;;zzz"
continueIfFileNotFound="false"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar,
content3rdParty.jar, content.jar"
completionJSMETHOD="uploadCompleted"
uploadedFromPath=""
completionUrl=""
workspacePath=""
docOperation="update"
failureJSMETHOD="uploadFailed"
debug="false"
failureUrl=""
uploadIfFileChanged="true"
targetType="NONE"
cabinets="wt/security/security.cab"
continueIfFileUnchanged="true"
oidString=""
contentDelim=";;qqq"
wt.context.locale="en_US"
delim=";;qqq"
uploadURL="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.co
ntent.ContentHttp/saveContent?wt.doc.WTDocument%3A656270"
removable="true"
fileName=""

```

```

attachments="FILE;;;qqqnewfile;;;qqqC:\TEMP\
DGadReq.doc;;;qqqADD;;;zzzURL;;;qqqnewURL;;;qqqwww.amazon.com;;;qqq;;;qqqADD;;;zzzF
ILE;;;qqqwt.content.ApplicationData:656293;;;qqqC:\TEMP\
MyDoc.doc;;;qqqREPLACE;;;zzzURL;;;qqqwt.content.URLData:656295;;;qqqhttp://www.askj
eeves.com;;;qqqAsk Jeeves
website;;;qqqREPLACE;;;zzzFILE;;;qqqwt.content.ApplicationData:656902;;;qqq;;;qqqRE
MOVE;;;zzzURL;;;qqqwt.content.URLData:656904;;;qqq;;;qqqREMOVE"
target=" "
><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

## The Download Applet

DownloadApplet is an "invisible" applet, that is, a 2-pixel by 2-pixel applet whose background color matches the page. It starts the download as soon as the page has loaded and the applet has initialized, using the values in the applet parameters.

This behavior makes DownloadApplet most appropriate for use in a "processing" page or popup rather than an interactive form page.

## Download of a Single File

Downloading a file through DownloadApplet requires two pieces of information: the filename to be download and the URL to download it from. These are indicated by setting the "filename" and "downloadURL" parameters. Depending on the user's preference for Download Operation Type, the file may be opened in application, saved to disk in a location chosen by the user, or the user may be prompted to choose between open-vs-save.

## Download of Multiple Files From one or More Windchill Objects

Files can be downloaded from multiple Windchill documents or multiple content items to a single destination with a single instance of DownloadApplet. This is done by setting the "filename" parameter to a concatenated list of filenames, and setting the "downloadURL" parameter to a concatenated list of downloadURLs corresponding in quantity and order to the concatenated filename values, and setting the "downloadQuantity" parameter to the quantity of files to be downloaded. Depending on the user's preference for Download Operation Type, the first file may be opened in application, saved to disk in a location chosen by the user, or the user may be prompted to choose between open-vs-save for that file. All other files will be downloaded to the same directory chosen for the first file, without further prompting.

## Configuring Download Behavior - User Preferences

The applet's prompt behavior and download location are determined by parameters that are typically set by the individual user's preferences.

The Download Operation Type preference (node=/wt/content key=downloadOpType) determines whether the downloaded file should be opened in

application, saved to disk or whether the user would like to be prompted to make that choice. The default is to prompt. For downloads of multiple files, this decision only applies to the first file downloaded (all other files will be saved to disk, in the same location).

The Default Local Directory preference (node=/wt/content key= workspacePath) determines where the file will be download upon open-in-application, or where the file chooser will open to upon save-to-disk. Users who have a particular directory where they like to keep all their Windchill files might like to set this preference to that directory. If no value is specified, or if the specified path does not exist on the local machine, then the Windows temp or Unix home directory will be used as specified by the environment variables of the local machine's operating system.

## Download Applet Parameters

Parameter Name	Sample Value	Description
java_codebase	http://machinename.ptcnet.ptc.com/Windchill/	Provided by taglib generator
type	application/x-java-applet;version=1.4	Provided by taglib generator
java_code	wt/clients/util/DownloadApplet	Applet location in codebase – never changes
cache_archive	"contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar, content3rdParty.jar, content.jar"	Java archives containing the classfiles used by this applet
java_archive	wt/security/security.jar	Never changes
cabinets	wt/security/security.cab	Never changes
cache_option	Plugin	Never changes
wt.context.locale	en_US	Locale of user's browser
downloadURL	http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redirectDownload/1026508screenshot.doc?u8&HttpOperationItem=wt.content.ApplicationData%3A294016&ContentHolder=wt.doc.WTDocument%3A294014"	URL that content will be downloaded from
filename	MyDoc.doc	Filename of file to be downloaded, used in prompt dialogs, can be changed during download.
url	http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.enterprise.URLProcessor/invokeAction?action=CheckOutDocCloseWindow&OID=VR%3Awt.doc.WTDocument%3A294013&formName=CheckOutDocumentWizard	URL to follow after processing is complete
downloadQuantity	3	(Optional) Number of different files and download sources, if downloading more than one file
delim	::;qqq	(Optional) Delimiter string used to separate filepaths if downloading more than one file

downloadOpType	ALWAYS_ASK, ALWAYS_SAVE, ALWAYS_OPEN	(Optional) Normally retrieved from user preference setting, determines whether user is prompted to open-in-app vs. save-to-disk
defaultPath	C:/temp/MyWindchillFiles/dummy.txt	(Optional) Normally retrieved from workspacePath user preference setting, determines where file is downloaded for open-in-app or where file browser is launched for save-to-disk
debug	true, false	(Optional) Defaults to false, change to true for Java Console output during processing

## Sample HTML for DownloadApplet

### Download (single file)

```
<OBJECT classid="clsid:CAFEEEFAC-0014-0002-0000-ABCDEFEDCBA" name="formApplet"
width="2" height="2" codebase="http://java.sun.com/products/plugin/autodl/jinstall-
1_4_2-windows-i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/http/DownloadApplet">
<param name="java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
<param name="java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="filename" value="1026508screenshot.doc">
<param name="url"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.enterp
rise.URLProcessor/invokeAction?action=CheckOutDocCloseWindow&OID=VR%3Awt.doc.WTDocu
ment%3A294013&formName=CheckOutDocumentWizard">
<param name="debug" value="FALSE">
<param name="delim" value=";;;qqq">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar,
lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="cabinets" value="wt/security/security.cab">
<param name="downloadOpType" value="ALWAYS_ASK">
<param name="cache_option" value="Plugin">
<param name="wt.context.locale" value="en_US">
<param name="downloadQuantity" value="1">
<param name="defaultPath" value="">
<param name="downloadURL"
value="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.mas
ter.RedirectDownload/redirectDownload/1026508screenshot.doc?u8&HttpOperationItem=wt
.content.ApplicationData%3A294016&ContentHolder=wt.doc.WTDocument%3A294014">
<param name="recordDelim" value=";;;zzz">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="2"
height="2" pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/http/DownloadApplet"
```

```

java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
filename="1026508screenshot.doc"
url="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.enterprise.URLProcessor/invokeAction?action=CheckOutDocCloseWindow&OID=VR%3Awt.doc.WTDocument%3A294013&formName=CheckOutDocumentWizard"
debug="FALSE"
delim=";;;qqq"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar, content3rdParty.jar, content.jar"
cabinets="wt/security/security.cab"
downloadOpType="ALWAYS_ASK"
cache_option="Plugin"
wt.context.locale="en_US"
downloadQuantity="1"
defaultPath=""
downloadURL="http://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redirectDownload/1026508screenshot.doc?u8&HttpOperationItem=wt.content.ApplicationData%3A294016&ContentHolder=wt.doc.WTDocument%3A294014"
recordDelim=";;;zzz"
><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```

## Download (multiple files)

```

<OBJECT classid="clsid:CAFEEFAC-0014-0002-0000-ABCDEFEDCBA" name="formApplet"
width="2" height="2" codebase="http://java.sun.com/products/plugin/autodl/jinstall-1_4_2-windows-i586.cab#Version=1,4,2">
<param name="java_code" value="wt/clients/util/http/DownloadApplet">
<param name="java_codebase" value="http://mecasey03d.ptcnet.ptc.com/Windchill/">
<param name="java_archive" value="wt/security/security.jar">
<param name="type" value="application/x-java-applet;version=1.4">
<param name="filename"
value=";;;qqqRequirements_Template.doc;;;qqqAgenda_Template.doc;;;qqqMemo_Template.doc;;;qqqMinutes_Template.doc;;;qqqMS_Project_Plan_Template.mpp;;;qqqPresentation_Template.ppt;;;qqq">
<param name="url" value="javascript:close()">
<param name="debug" value="FALSE">
<param name="delim" value=";;;qqq">
<param name="cache_archive" value="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar, content3rdParty.jar, content.jar">
<param name="cabinets" value="wt/security/security.cab">
<param name="downloadOpType" value="ALWAYS_ASK">
<param name="cache_option" value="Plugin">
<param name="wt.context.locale" value="en_GB">
<param name="downloadQuantity" value="6">
<param name="defaultPath" value="">
<param name="downloadURL"
value=";;;qqqhttp://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redirectDownload/Requirements_Template.doc?u8&HttpOperationItem=wt.content.ApplicationData%3A295941&ContentHolder=wt.doc.WTDocument%3A295939;;;qqqhttp://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redirectDownload/Agenda_Template.doc?u8&HttpOperationItem=wt.content.ApplicationData%3A294345&ContentHolder=wt.doc.WTDocument%3A294335;;;qqqhttp:

```

```

/mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.Redirect
Download/redirectDownload/Memo_Template.doc?u8&HttpOperationItem=wt.content.Applica
tionData%3A294363&ContentHolder=wt.doc.WTDocument%3A294353;;;qqqhttp://mecasey03d.p
tcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redir
ectDownload/Minutes_Template.doc?u8&HttpOperationItem=wt.content.ApplicationData%3A
294381&ContentHolder=wt.doc.WTDocument%3A294371;;;qqqhttp://mecasey03d.ptcnet.ptc.c
om/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redirectDownload
/MS_Project_Plan_Template.mpp?u8&HttpOperationItem=wt.content.ApplicationData%3A294
399&ContentHolder=wt.doc.WTDocument%3A294389;;;qqqhttp://mecasey03d.ptcnet.ptc.com/
Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redirectDownload/Pr
esentation_Template.ppt?u8&HttpOperationItem=wt.content.ApplicationData%3A295917&Co
ntentHolder=wt.doc.WTDocument%3A295907;;;qqq">
<param name="recordDelim" value=";;;zzz">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.4" name="formApplet" width="2"
height="2" pluginspage="http://java.sun.com/getjava/download.html"
java_code="wt/clients/util/http/DownloadApplet"
java_codebase="http://mecasey03d.ptcnet.ptc.com/Windchill/"
java_archive="wt/security/security.jar"
filename=";;;qqqRequirements_Template.doc;;;qqqAgenda_Template.doc;;;qqqMemo_Templa
te.doc;;;qqqMinutes_Template.doc;;;qqqMS_Project_Plan_Template.mpp;;;qqqPresentatio
n_Template.ppt;;;qqq"
url="javascript:close()"
debug="FALSE"
delim=";;;qqq"
cache_archive="contentCust.jar, contentDSU.jar, contentFCS.jar, lib/HTTPClient.jar,
content3rdParty.jar, content.jar"
cabinets="wt/security/security.cab"
downloadOpType="ALWAYS_ASK"
cache_option="Plugin"
wt.context.locale="en_GB"
downloadQuantity="6"
defaultPath=""
downloadURL=";;;qqqhttp://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuth
GW/wt.fv.master.RedirectDownload/redirectDownload/Requirements_Template.doc?u8&Http
OperationItem=wt.content.ApplicationData%3A295941&ContentHolder=wt.doc.WTDocument%3
A295939;;;qqqhttp://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.
fv.master.RedirectDownload/redirectDownload/Agenda_Template.doc?u8&HttpOperationIte
m=wt.content.ApplicationData%3A294345&ContentHolder=wt.doc.WTDocument%3A294335;;;qq
qhttp://mecasey03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.Re
directDownload/redirectDownload/Memo_Template.doc?u8&HttpOperationItem=wt.content.A
pplicationData%3A294363&ContentHolder=wt.doc.WTDocument%3A294353;;;qqqhttp://mecase
y03d.ptcnet.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload
/redirectDownload/Minutes_Template.doc?u8&HttpOperationItem=wt.content.ApplicationD
ata%3A294381&ContentHolder=wt.doc.WTDocument%3A294371;;;qqqhttp://mecasey03d.ptcnet
.ptc.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redirectDo
wnload/MS_Project_Plan_Template.mpp?u8&HttpOperationItem=wt.content.ApplicationData
%3A294399&ContentHolder=wt.doc.WTDocument%3A294389;;;qqqhttp://mecasey03d.ptcnet.pt
c.com/Windchill/servlet/WindchillAuthGW/wt.fv.master.RedirectDownload/redirectDownl
oad/Presentation_Template.ppt?u8&HttpOperationItem=wt.content.ApplicationData%3A295
917&ContentHolder=wt.doc.WTDocument%3A295907;;;qqq"
recordDelim=";;;zzz"
><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

```





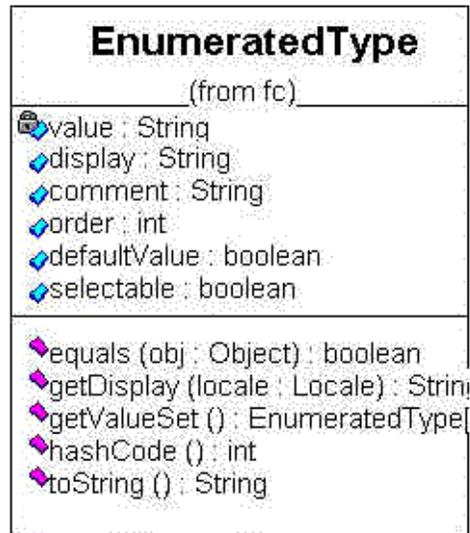
# A

## Enumerated Types

<b>Topic</b>	<b>Page</b>
The EnumeratedType Class.....	A-2
Creating an EnumeratedType Subclass.....	A-3
Editing the Resource Info for an Enumerated Type.....	A-7
Localizing an Enumerated Type.....	A-10
Extending an Enumerated Type .....	A-11
The Enumerated Type Customization Utility .....	A-12
GUI Usage of an Enumerated Type .....	A-13

## The EnumeratedType Class

EnumeratedType represents a type whose possible values are constrained to a set (as defined in a resource).



### value

The internal value, which is persisted.

### display

The localizable display text.

### comment

An optional comment describing the value.

### order

Provides an explicit sort order for the value.

### defaultValue

Specifies the value that is the default value for the type.

### selectable

Specifies if the value should be allowed to be selected.

The constructors for EnumeratedTypes are protected so that instances can only be constructed internally. The data needed for construction is obtained from a resource and used to construct the instances in the static initializer.

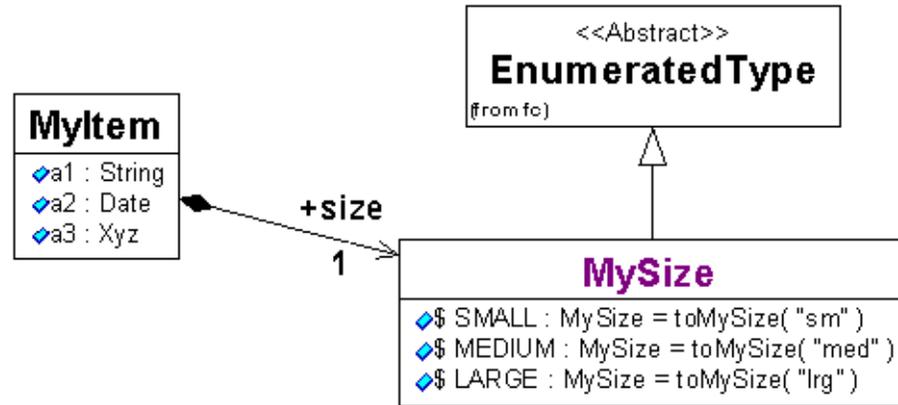
getValueSet() returns the set of possible values for the class, where the possible values are instances of the class.

toString() returns the internal value, which will be persisted. This follows the pattern of primitive wrappers provided by Sun. This means toString() is not available for generic use by GUI components; they must use getDisplay().

## Creating an EnumeratedType Subclass

To create an EnumeratedType subclass, perform the following steps:

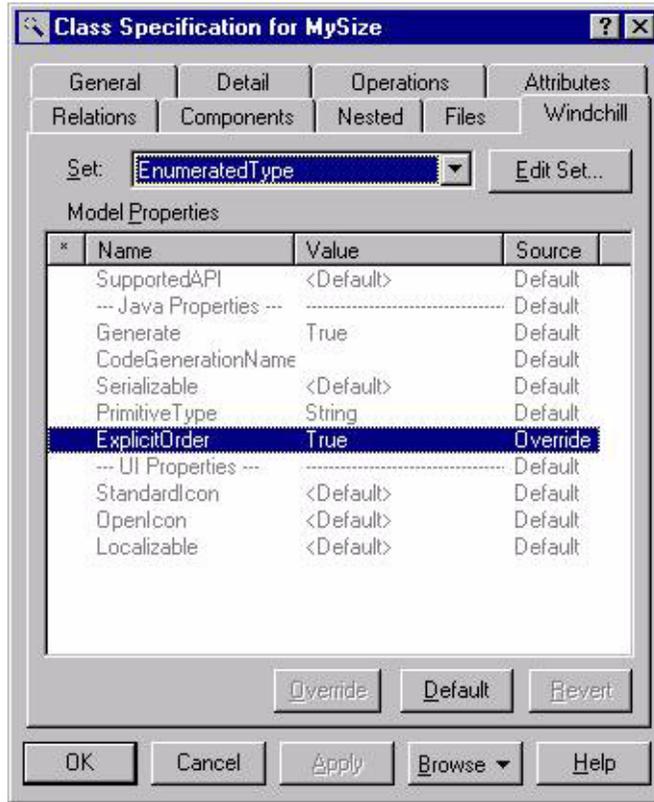
1. Model a class that extends wt.fc.EnumeratedType.



Select the EnumeratedType property set on the **Windchill** tab of the Class Specification. This property set provides just the properties that apply to EnumeratedType classes, and changes the default values of certain properties, such as PrimitiveType to String.

If the values are to be ordered explicitly, set the ExplicitOrder property to True. Otherwise, the values will be ordered alphabetically at runtime, using the locale-specific display.

The Localizable property under the UI Properties section refers to the name of the class being localizable, not the display values.



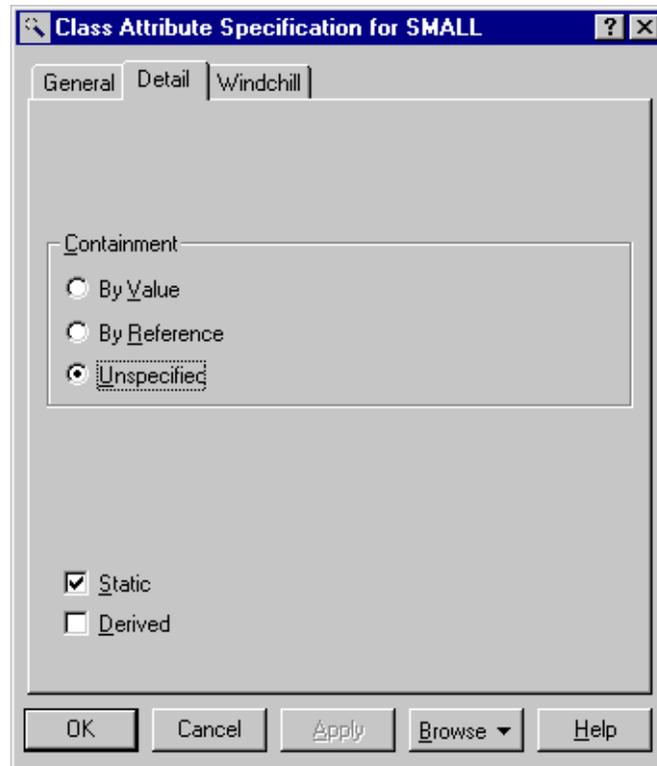
2. Generating the EnumeratedType also generates its companion resource info file. For MySize, it will be named MySizeRB.rbInfo.

If the wt.resource.updateLocales property of tools.properties is set to true, any locale-specific versions of the rbInfo file will also be updated, indicating which entries need to be translated.

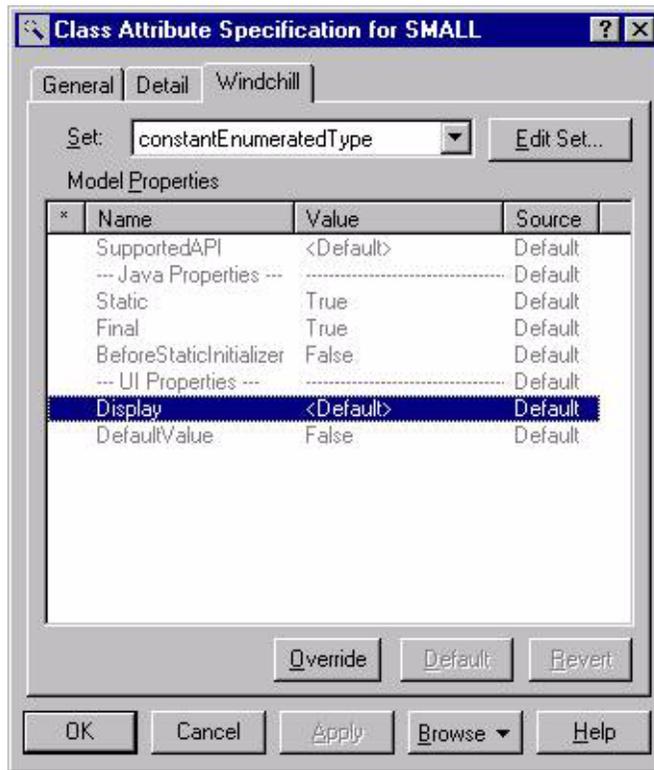
3. Optionally, model any desired programmatic constants. For example:

```
SMALL : MySize = toMySize( "Sm" )
```

Modeled constants are Static and Final. A value definition for each modeled constant will be generated in the resource info file.



Select the constantEnumeratedType property set on the Windchill tab of the Attribute Specification. This property set provides just the properties that apply to EnumeratedType constants, and changes the default values of certain properties, such as BeforeStaticInitializer to False. If the constant represents the default value for the EnumeratedType, set the DefaultValue property to True. To explicitly define the display name to generate into the resource info file, set the Display property. Otherwise, the generated display name will be based on the initial value of the constant.



## Editing the Resource Info for an Enumerated Type

The name of the generated resource info file is **RB.rbInfo**, with the simple package name prepended, for example, `MySizeRB.rbInfo`. The following sections describe the resource info files that are generated.

### Header

Each resource info file contains the following lines that define certain file level information:

```
ResourceInfo.class=wt.tools.resource.EnumResourceInfo
    ResourceInfo.customizable=true
    ResourceInfo.deprecated=false
```

The first line classifies the resource info and should never be changed. The values of the second two lines can be changed by the owner of the package, if the file should not be customized and if the file is deprecated.

### Resource Entry Format

This section lists the valid keys for each resource entry:

```
# Entry Format (values equal to default value are not included)
    # <key>.value=
    # <key>.comment=
    # <key>.argComment<n>=
    # <key>.constant=
    # <key>.customizable=
    # <key>.deprecated=
    # <key>.abbreviatedDisplay=
    # <key>.fullDisplay=
    # <key>.shortDescription=
    # <key>.longDescription=
    # <key>.order=                                (Explicit sort order
    for the value, unused for alpha ordering.)
    # <key>.defaultValue=                          (Specifies the value is
    the default value for the Enumerated Type.)
    # <key>.selectable=                            (Specifies if the value
    should be allowed to be selected.)
```

The key `<key>.value` is the only key that is required to have a value defined. This key, along with the two Display values (abbreviatedDisplay and fullDisplay), the two Description values (shortDescription and longDescription), order, and defaultValue are the only keys that are localizable.

The `<key>.constant` value is unused for these EnumeratedType entries, but is used for other types of resource info files.

## Resource Entry Contents

The following are examples of the entries that are generated:

```
lrg.value=lrg
lrg.order=30

med.value=med
med.order=20

sm.value=sm
sm.order=10
```

These default values are generated only once, so that the package owner can override the default values, if desired. If the class's ExplicitOrder property is set to True, the orders are controlled by the generator, and should not be edited directly in the resource info file. If one of the constants has DefaultValue set to True, it is controlled by the generator. For example, the value for the "sm" value could be changed as follows:

```
sm.value=Small
sm.shortDescription=Small Comment
sm.order=10
sm.defaultValue=true
```

## Building Runtime Resources

Windchill provides a utility to build the runtime resource for the .rbInfo files.

- To build the runtime resources into the codebase for a particular RB file, use the following command:

**ResourceBuild** *<package.name>*

For example:

```
ResourceBuild wt.example.MySizeRB
```

- To build the runtime resource into the codebase for all the resource info files for a particular directory, use the following command:

**ResourceBuild** *<directory\relative\to\src>*

For example:

```
ResourceBuild wt\example
```

The resulting resource file is named `<name>.RB.ser`, which is a serialized instance of `SerializedResourceBundle`. For example, `src\wt\example\MySizeRB.rbInfo` will build to `codebase\wt\example\MySizeRB.RB.ser`.

To verify the values stored in a resource bundle, a verification utility is provided by the `EnumeratedType` base class. A batch file, `enumVerify.bat`, can be used to invoke this verification, as follows:

```
enumVerify <fully.qualified.EnumClassname>[<language>][<country>]  
[<variant>]
```

The following are examples of usage:

```
enumVerify wt.lifecycle.State  
enumVerify wt.lifecycle.State fr  
enumVerify wt.lifecycle.State fr CA
```

For information on locales, and codes for languages and countries, see the Javadoc for `java.util.Locale`.

## Localizing an Enumerated Type

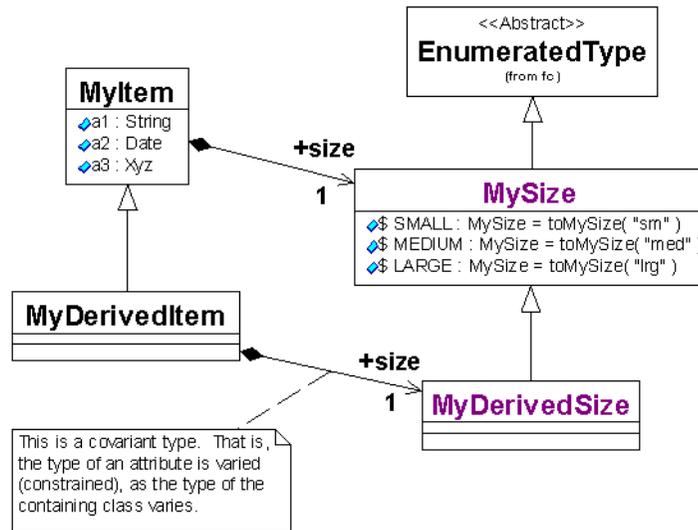
To have a locale-specific set of display values for an enumerated type, a locale-specific version of the enumerated type's resource bundle can be created. To do so, perform the following steps:

1. Make a copy of the standard resource bundle for the enumerated type, giving it a file name with the appropriate locale suffix. For example, to create a French version of the resource bundle for MySize, copy the MySizeRB.rbInfo bundle and name it MySizeRB\_fr.rbInfo.
2. Do not change the keys of entries being localized; they must match those in the standard bundle.
3. Edit the display and description fields with the appropriate translation.
4. Change the order and defaultValue fields if desired; it is not likely to be necessary.
5. Do not change the selectable field.

After creating the resource info file for a specific locale and translating it, build the runtime resource from it. Then verify the results of your localization by using the verification utility mentioned earlier.

## Extending an Enumerated Type

An enumerated type can be extended, and using it in a constraining, or covariant, manner is supported. For example, `MySize` can be extended by `MyDerivedSize`, and it can be used as indicated by `MyDerivedItem`, as shown in the following figure.



The one caveat with using extended `EnumeratedType` instances is that, if concrete types are used in the model, they are the only types that can be read back from the database. Using the example in the preceding figure, this means that other subclasses of `MySize` can be assigned to the `size` attribute of `MyItem`, and they can be stored in the database, but they can be read out only as instances of the types that are modeled.

This limitation would not apply if `MySize` were an abstract class. When an abstract class is modeled, the runtime type information is stored in the database along with the instance information. Therefore, the exact type and instance can be reconstructed when reading it back from the database.

# The Enumerated Type Customization Utility

The Enumerated Type Customization Utility allows you to add or edit values of an enumerated type. This utility should be used rather than directly creating customized resource info files. The utility will create, or update, files in a parallel directory structure defined by the wt.generation.custom.dir property, which defaults to \$(wt.home)/wtCustom. The resource info files provided with Windchill should never be changed.



The remainder of this section describes how to start the utility. For specific instructions on its usage, see the online help available when you start the utility.

## Starting the Utility

In the `tools.properties` file, the `wt.resource` and `wt.clients.tools.enumtype` properties (described in `properties.html`) are used to configure the utility for your environment. Most of these properties need not be set, since appropriate defaults are provided.

To start the Enumerated Type Customization Utility, open a console prompt and enter the following script name:

```
enumCustomize
```

To see the changes you have made after using the utility, perform the following steps:

1. Restart the method server.
2. Rebuild the `wt.jar` file so clients can access the new values.
3. Restart any Java clients. (HTML clients access the new values as soon as their pages are refreshed from the server.)

To test new values that impact GUIs, you may want to run in an environment without a `wt.jar` file so you can see the changes without downloading the new file.

## GUI Usage of an Enumerated Type

The localizable display field for an enumerated type can be displayed by using `getDisplay()` or `getDisplay(locale)`.

To use enumerated types in GUI drop-down list selections, `wt.clients.util.EnumeratedChoice` is provided as a generic control for enumerated types. The following is a simple example of using the control:

```
// construct and initialize a ChoiceBox for an EnumeratedType
EnumeratedChoice mySizeChoice = new EnumeratedChoice();
mySizeChoice.setEnumeratedTypeClassName( "wt.example.MySize" );

// set current choice selection to the current value
mySizeChoice.setSelectedEnumeratedType( myItem.getSize() );

... user makes selection

// obtain the chosen selection
MySize selection =
    (MySize)mySizeChoice.getSelectedEnumeratedType();
```



# B

## Extendable Classes in the Windchill Supported API

This appendix lists the extendable classes in the Windchill supported API. It serves as a roadmap to help guide developers when initiating modeling activities. Further information about these classes can be found throughout this manual, the *Windchill Application Developer's Guide*, and in the Windchill Javadoc.

<b>Topic</b>	<b>Page</b>
PDM Business Information Classes .....	B-2
Enterprise Business Information Classes .....	B-2
Windchill Services .....	B-2
Foundation Classes .....	B-3
PDM Auxiliary Business Information Classes.....	B-3
Business Logic Classes .....	B-3
Server Development Classes .....	B-4
Client Development Classes.....	B-5

## PDM Business Information Classes

Following are the starting points for extending PDM objects (part, document, and change objects). If you want to extend the behavior of the Windchill out-of-the-box PDM objects, start by extending these classes. For an overview of these classes, see the *Windchill Application Developer's Guide* chapter on the enterprise layer.

```
wt.change2.WTAnalysisActivity
wt.change2.WTChangeActivity2
wt.change2.WTChangeInvestigation
wt.change2.WTChangeIssue
wt.change2.WTChangeProposal
wt.change2.WTChangeRequest2
wt.doc.WTDocument
wt.doc.WTDocumentMaster
wt.part.WTPart
wt.part.WTPartMaster
```

## Enterprise Business Information Classes

To create new business classes (other than the PDM classes described above), the following enterprise classes provide a good base from which to start:

```
wt.enterprise.FolderResident
wt.enterprise.IteratedFolderResident
wt.enterprise.Managed
wt.enterprise.RevisionControlled
wt.enterprise.Simple
```

## Windchill Services

The following extendable classes represent the Windchill plug-and-play interfaces. For more information, see the *Windchill Application Developer's Guide* chapter on Windchill services.

```
wt.change2.Changeable2
wt.content.ContentHolder
wt.content.FormatContentHolder
wt.folder.Foldered
wt.folder.IteratedFoldered
wt.index.Indexable
wt.locks.Lockable
wt.ownership.Ownable
wt.vc.baseline.Baselineable
wt.vc.Iterated
wt.vc.Mastered
wt.vc.Versioned
wt.vc.views.ViewManageable
wt.vc.wip.Workable
```

## Foundation Classes

For an overview, see the *Windchill Application Developer's Guide* chapter on modeling business objects, specifically the section within that chapter on Windchill foundation abstractions.

```
wt.fc.BinaryLink
wt.fc.IdentificationObject
wt.fc.Identified
wt.fc.Item
wt.fc.Link
wt.fc.NetFactor
wt.fc.ObjectMappable
wt.fc.ObjectReference
wt.fc.ObjectToObjectLink
wt.fc.Persistable
wt.fc.QueryKey
wt.fc.SemanticKey
wt.fc.UniquelyIdentified
wt.fc.WTObject
wt.fc.WTReference
```

## PDM Auxiliary Business Information Classes

The following extendable classes provide some of the basic functionality of the PDM business information classes.

```
wt.vc.ObjectToVersionLink
wt.vc.struct.IteratedDescribeLink
wt.vc.struct.IteratedReferenceLink
wt.vc.struct.IteratedUsageLink
wt.VersionToVersionLink
```

## Business Logic Classes

Following are extendable business logic classes:

```
wt.eff.DateEff
wt.eff.DisplayIdentificationEffConfigurationItemDelegate
wt.eff.DisplayIdentificationProductInstanceDelegate
wt.eff.Eff
wt.eff.EffConfigurationItem
wt.eff.EffContext
wt.eff.EffGroupRangeDelegate
wt.eff.EffManagedVersion
wt.eff.EffRange
wt.eff.IncorporationDate
wt.eff.LeftFilledStringEff
wt.eff.ProductSolution
wt.eff.StringEff
wt.eff.ValidEffsDelegate
wt.effectivity.DatedEffectivity
wt.effectivity.Effectivity
wt.effectivity.UnitEffectivity
wt.part.BOMHierarchyVisitor
```

wt.part.BOMPartsListVisitor  
wt.part.BOMTemplateProcessor  
wt.part.HtmlHeadingHierarchyVisitor  
wt.part.HtmlNumberedHierarchyVisitor  
wt.part.HtmlPreformattedHierarchyVisitor  
wt.part.HtmlPreformattedPartsListVisitor  
wt.part.HtmlTabularPartsListVisitor  
wt.series.HarvardSeries  
wt.series.IntegerSeries  
wt.series.MulticharacterSeries  
wt.series.MultilevelSeries  
wt.series.Series  
wt.vc.baseline.Baseline  
wt.vc.baseline.BaselineVisitor  
wt.vc.baseline.ManagedBaseline  
wt.vc.config.ConfigSpec  
wt.visitor.BasicNodeExpander  
wt.visitor.CompositeVisitor  
wt.visitor.ConfigSpecNodeExpander  
wt.visitor.Expander  
wt.visitor.Navigator  
wt.visitor.NodeExpander  
wt.visitor.NodeExpandInfo  
wt.visitor.RoleExpandInfo  
wt.visitor.TextOutputVisitor  
wt.visitor.Visitor  
wt.visitor.VisitorAdapter  
wt.visitor.Walker

## Server Development Classes

Following are extendable server development classes:

wt.cache.CacheManager  
wt.doc.DocumentContentProcessing  
wt.doc.DocumentDelegate  
wt.doc.WTDocumentDelegate  
wt.events.KeyedEvent  
wt.events.KeyedEventBranch  
wt.events.KeyedEventListener  
wt.events.KeyedEventListenerAdapter  
wt.feedback.ProgressCountFeedback  
wt.feedback.StatusFeedback  
wt.feedback.WTContextUpdate  
wt.httpgw.HTTPAuthentication  
wt.index.IndexDelegate  
wt.pom.TransactionListener  
wt.util.CollationKeyFactory  
wt.util.WTThread

## Client Development Classes

Following are extendable client development classes:

```
wt.clients.beans.AssociationsLogic
wt.clients.beans.EffectivityTaskLogic
wt.clients.beans.explorer.WTBusinessObject
wt.clients.beans.selectors.ComponentDelegate
wt.clients.change2.WTChangeIssueTaskDelegate
wt.clients.change2.WTChangeOrder2TaskDelegate
wt.clients.change2.WTChangeRequest2TaskDelegate
wt.clients.doc.WTDocumentTaskDelegate
wt.clients.effectivity.ConfigurationItemTaskDelegate
wt.clients.folder.SubFolderTaskDelegate
wt.clients.homepage.WindchillHome
wt.clients.prodmgmt.HelperPanel
wt.clients.prodmgmt.PartItem
wt.clients.prodmgmt.PartMasterItem
wt.clients.util.ReferenceHolder
wt.clients.util.TaskDelegateException
wt.clients.vc.AlreadyCheckedOutException
wt.clients.vc.CheckedOutByOtherException
wt.clients.vc.CheckInOutException
wt.clients.vc.NotCheckedOutException
wt.identity.DisplayIdentification
wt.identity.DisplayIdentificationDelegate
wt.identity.DisplayIdentificationPersistableDelegate
wt.identity.DisplayIdentificationStandardDelegate
wt.identity.DisplayIdentificationStandardVersionedDelegate
wt.identity.DisplayIdentificationVersionedDelegate
wt.identity.DisplayIdentity
wt.identity.StandardDisplayIdentity
wt.identity.VersionedDisplayIdentity
wt.query.report.MacroProcessor
wt.util.LocalizableMessage
```